DESIGN ROBUSTNESS ANALYSIS OF NEUROMORPHIC CIRCUITS

A Thesis

by

AHMAD ABDEL MAJID SULEIMAN BASHAIREH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Committee Members,	Gwan Choi
	Eun Jun Kim
Head of Department,	Chanan Singh

May 2014

Major Subject: Electrical Engineering

ABSTRACT

Conventional Von Neumann architecture faces significant challenges as the device dimensions are scaled down. Power consumption and device reliability have become major concerns. Therefore, new computational paradigms are being proposed to overcome these challenges. Brain-inspired computing has emerged as a promising direction in that regard. Its massive-parallelism, potential for scalability, and power efficiency make it attractive. In addition, neuromorphic computing has shown better performance in complex tasks such as pattern recognition.

Few attempts have been made to investigate the effect of silicon failures beyond the circuit level. In this thesis, a method is proposed to evaluate the impact of process and environmental variations on the overall performance of biologically inspired spiking neural networks. In this method, transistor-level and behavioral level analysis are carried out. Then, the results of the transistor-level simulation is mapped to the application layer to determine effect of variability on the performance of the system.

Monte Carlo analysis of a brain-inspired digital neuromorphic circuit in the presence of voltage, and temperature (PVT) variations is performed. A commercial 90nm technology process is utilized to synthesize and simulate the design. The functionality of the circuit is demonstrated through a behavioral model of a neural network that implements a character recognition system. Errors are injected in the network to obtain its fault resilience characteristics. The result from PVT variations analysis are projected into a behavioral model to estimate the effect of the circuit failures on the operation of the neural network. Furthermore, the influence of key parameters on the system's performance is examined. These are the supply voltage, at the cir-

cuit level, and the structure, at the application level. The experimental results have demonstrated the robustness of the networks with respect to the targeted variation effects.

DEDICATION

To my family

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Peng Li, for his advice, patience, and support throughout this research. Also, I want to thank my committee members, Dr. Gwan Choi and Dr. Eun Jung Kim, for their insightful comments and suggestions on this work.

I would like to thank our research group members for their help and constructive discussions. I am specially thankful to Yongtae Kim, for providing the neuromorphic circuit design, and Yong Zhang, for providing the behavioral simulator.

I am grateful to my friends for being there whenever I needed them and making my stay in College Station a pleasant experience. I wish to thank Mohammad Albaba for his help in reviewing and editing this thesis.

I cannot find the words to express my gratitude to my family. Without their unconditional love and support I would no be where I am today. I owe them everything. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Biological systems have the ability to perform complex operations, such as pattern recognition, at low power, low speed, and small footprint. The same functionality requires much more power and space if it were to be performed by the conventional Von Neumann architecture. Although they operate at low speed, they are able to achieve high throughput through parallel processing based on a huge number of processing units. In general, biological neurons are a million time slower than silicon gates. Today's integrated chips operate at a clock period in the range of nanoseconds (10^{-9} s) while neural events may happen in the millisecond (10^{-3} s) range. Moreover, the brain consumes approximately 10^{-16} J per operation per second, whereas the traditional computer requires an energy level of about 10^{-6} J per operation per second [11, 9, 5].

This has led to an increasing interest in building bio-inspired systems to leverage such advantages. Specifically, brain-inspired architecture is an emerging field that has attracted wide research interest. This is in part because the brain shows error resilience and fault tolerance which are important in VLSI systems. On one hand, this can be utilized to enhance the performance with the presence of process variation and device reliability issues. On the other hand, a 100% precision is not required for many cognitive computing applications, which allows for approximate computing, and hence low-power consumption. In addition, these systems may provide an answer to the challenges facing traditional architecture stemming from shrinking device dimensions, such as process variation, device reliability, and power consumption.

This research aims at establishing an approach to examine the robustness of neuromorphic circuits. While the error resilience of the brain-inspired designs is rea-

sonably established at the software level, there are few in-depth attempts at demonstrating that an actual hardware implementation of such systems would be tolerant to silicon failures [15]. Therefore, a method will be proposed to determine the effect of process and environmental variations on the performance of a digital neuromorphic circuit at the application level, hence, providing a comprehensive and realistic analysis of the resilience of such systems. It will be shown that silicon neurons, implemented in digital circuits, perform well in the presence of large process, voltage, and temperature (PVT) variations.

The structure of this thesis is as follows: the remainder of this introductory chapter introduces the concept of brain-inspired computing, then the circuit and application under investigation. Chapter 2 describes the methodology of this research, opening with a background and related work, and moving to the research approach afterwards. Experimental setting and results are presented in chapter 3. Finally, conclusions and future work are discussed.

1.1 Artificial Neural Networks

Artificial neural networks (ANN) are computational models inspired by the human brain. It is a large network of connected neurons that process data in a similar manner to the nervous system. The neurons are the processing units in the network. They are connected through synapses, which have weights associated with them that reflects the strength of a connection. In addition to providing connectivity, the synapse is the learning element in ANNs. In this structure, the sending neuron and the receiving neuron are called pre-synaptic and post-synaptic neurons, respectively. When a neuron receives an input, it calculates the sum of the input signals, each weighted by its synapse strength. After that, the output is calculated based on an activation function applied to the sum. This process is shown Figure 1.1 and can be

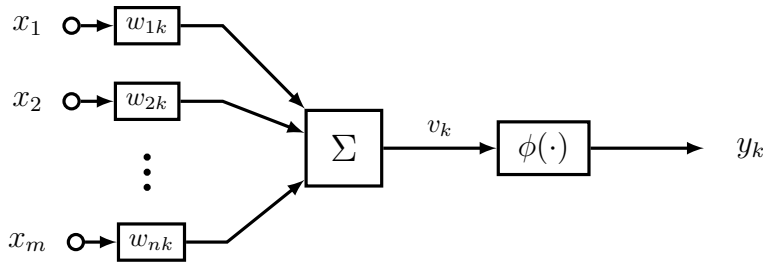


Figure 1.1: ANN neuron

mathematically formulated as:

$$v_k = \sum_{i=1}^{n} w_{ik} x_i \tag{1.1}$$

$$y_k = \phi(v_k) \tag{1.2}$$

where n is the number of pre-synaptic neurons connected to the current neuron (k), w_{ik} is the weight for the synapse between neuron i and k, x_i is the signal transmitted from neuron i to k, y_k is the output of neuron k, and $\phi(\cdot)$ is the activation function [9, 5].

It is worth mentioning that ANNs are structured into layres: input, output, and hidden layers. The input layer receives external stimuli in the form of activation pattern and pass it the next layer. The hidden layer adds flexibility to the network and enables high-order computation. The final layer is the output layer where the overall response of the ANN is presented. Moreover, ANN are not restricted to feedforward architecture, rather a feedback loop may be present in the network. Such network is known as a recurrent network, which behave like a sequential logic circuit [9].

Learning in ANN refers to the process where the weights of the synapses are modified to perform particular function. Learning algorithms can be classified into

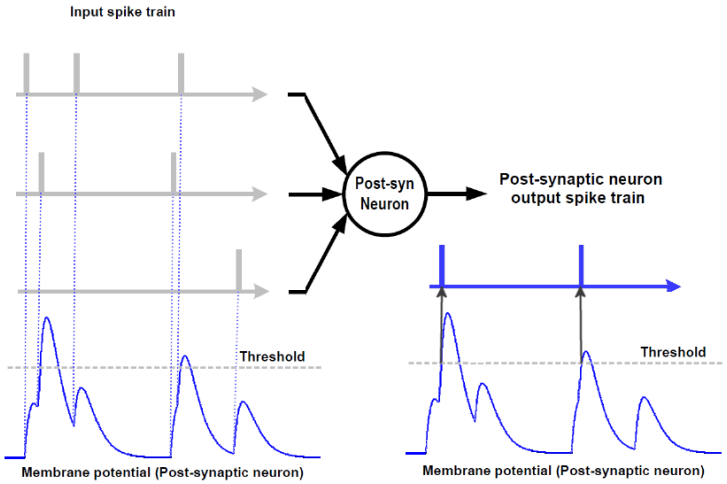


Figure 1.2: SNN neuron

two main categories: supervised and unsupervised [9, 6]. In supervised learning, the input and the desired output are provided to the network. During such process, the ANN tries to match the required output as close as possible through synapses' weight adjustment. In contrast, only the input is given in unsupervised learning. This type of learning utilizes the properties and correlations of the inputs and tries to find patterns and classify them. The neurons compete among themselves for activation while learning. This scheme is referred to as winner-take-all (WTA) because, ideally, one output neuron would be activated in response to a specific input.

The implemented ANN in this research falls within the spiking neural networks (SNN) category. This model incorporates the timing of spikes, which is more comprehensive and accurate in depicting the actual biological system. Rather than using the real-valued amplitude of a spike as in conventional ANN, SNN uses the existence of a spike and the relative timing between different spikes. Moreover, SNN assumes that all the information is encoded in the timing of the spike and the amplitude has no information associated with it.

In such system, the neuron receives a spike train and responds by generating another train of spikes as an output. With each input spike, the potential of the receiving neuron, which represents the internal state of the neuron, changes, increasing, in the case of excitatory pre-synaptic neuron, or decreasing, in the case of inhibitory pre-synaptic neuron. The neuron integrates the input spikes temporally, and in the event that the potential surpasses a threshold it fires a spike. Figure 1.2 gives an example of the operation of a spiking neuron [9]. In this example, all the pre-synaptic neurons are excitatory [9].

As with the conventional ANN, the synaptic weights reflect the strength of a connection between two neurons. Moreover, the weights are updated according to the timing of spikes, an input spike coming after an output spike will weaken the connection, and it is strengthened if the input spike precedes the output one. This scheme is called spike-timing-dependent plasticity (STDP) [9, 14]. Figure 1.3 shows the basic principle of STDP, which can be mathematically expressed as:

$$\Delta t = t_{post} - t_{pre} \tag{1.3}$$

$$\Delta w = W(\Delta t) \tag{1.4}$$

$$w = w + \Delta w \tag{1.5}$$

where t_{post} and t_{pre} are the firing time of pre-synaptic and post-synaptic neurons, respectively, $W(\cdot)$ is the STDP learning function, and w is the synaptic weight.

1.2 Digital Neuromorphic Processor Architecture

Figure 1.4 shows the overall block diagram of the digital neuromorphic architecture [10]. It implements a recurrent STDP neural network of 256 neurons and 256x256 synapses. This circuit has four main components: a synapse unit (SU), a

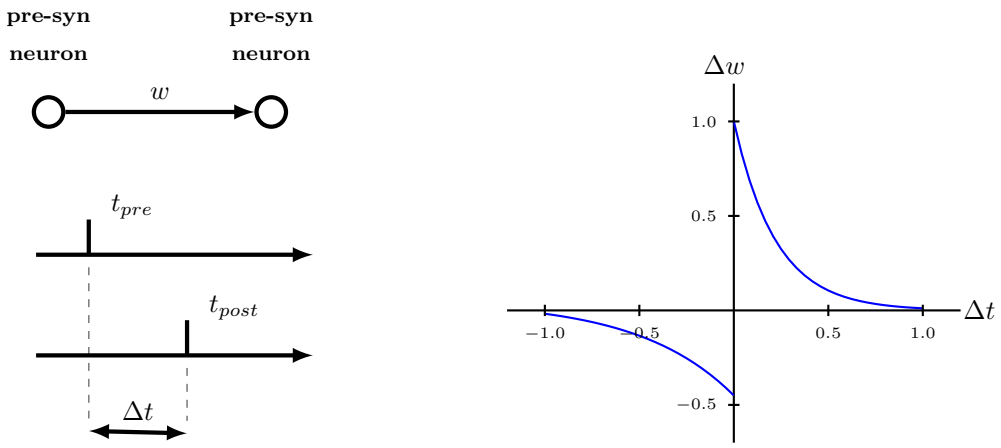


Figure 1.3: Spike-timing-dependent plasticity

learning unit (LU), a neuron unit (NU), and a top controller.

The SU is essentially a memory array that stores the synaptic weights. It is implemented using memristor devices. The LU has 256 learning elements (LE), each is associated with a neuron, which perform the STDP functionality. The NU also has 256 elements (NE) which follow the leaky integrate-and-fire (LIF) model. The top controller utilizes a synchronous design to manage the operation of the whole circuit. Each biological step takes multiple clock cycles [9]. Moreover, the signals at the input and output of the circuit are encoded in spikes as in a biological system.

The circuit operates in the following steps: it receives all the input spikes, then it is relayed to the output neurons which implement the following LIF dynamics:

$$V_i[t] = V_i[t-1] + K_{SYN} \sum_{j=1}^{m} w_{ji} S_j[t-1] + K_{EXT} E_i[t-1] - V_{LEAK} \tag{1.6}$$

where V_i is the membrane potential of neuron i, K_{SYN} is the synaptic weight parameter, M is the number of pre-synaptic neurons, w_{ji} is synaptic weight between neurons j and i, S_j is the activity bit which reflects if neuron j fired, K_{EXT} is the

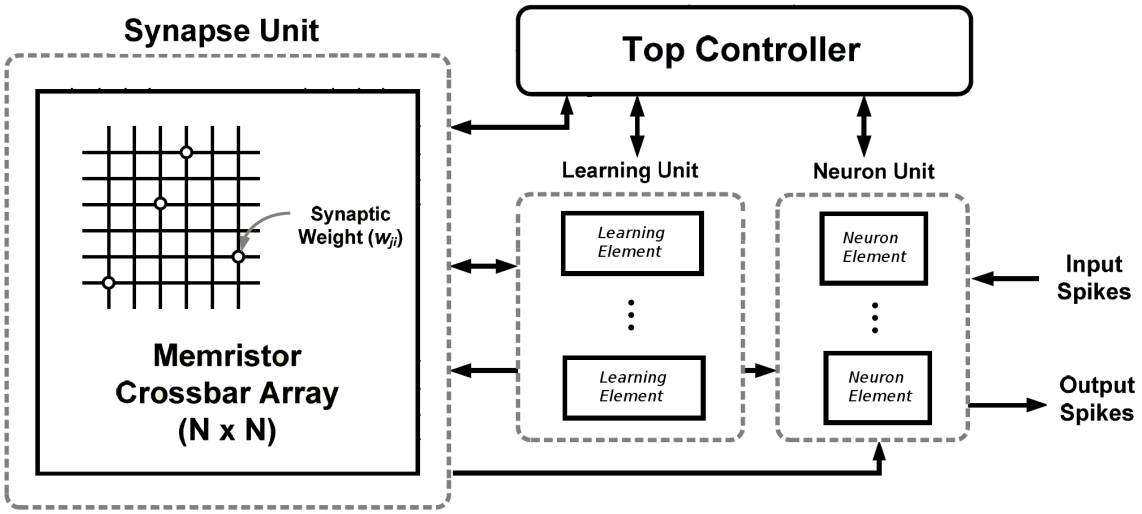


Figure 1.4: Block diagram of the digital neuromorphic circuit

external input parameter, E_i is the activity bit for the input spike, and V_{LEAK} is the leaky parameter. If V_i exceeds the threshold voltage, it fires, its S_i is set to one, and its V_i is reset. This is repeated for all the neurons in the circuit.

The final stage of the circuit operation is the learning, where the synaptic weights are updated using the STDP rule. Table 1.1 summarizes the implementation of the neuromorphic circuit [9].

1.3 Application to Neuromorphic Circuit to Character Recognition

An implementation of SNN at a behavioral level is used to demonstrate the functionality of the neuromorphic circuit. This is done through a model, rather than simulating an application directly on the circuit, because the complexity of the simulation at a lower level requires huge processing time. Specifically, in such an application which requires long training time, transistor level simulation is practically infeasible. Nonetheless, the model closely captures the dynamics of the of neuromorphic circuit, especially its main elements, the LIF neuron and the STDP learning

Table 1.1: Summary of neuromorphic circuit parameters

Technology	**90nm**
Supply Voltage	1.2 V
Neuron Model	LIF
Learning Rule	STDP
Synapse device	Memrsitor
Power Dissipation	6.45 mW
Area	1.86 mm^2
No. of Neurons	256
No. of Synapses	65536

units.

The implemented SNN is a two-layer network with an input and output layers and performs unsupervised character recognition [9]. It consists of 1066 neurons, 1060 excitatory and 6 inhibitory. The input layer contains 1024 excitatory neurons and 6 inhibitory neurons. The excitatory neurons receive the input pattern, a 32x32 pixel binary image, and project it on the output layer through the synapse connections. In addition, the excitatory input neurons are connected to inhibitory neurons that provide feedback to modulate the firing frequency of the excitatory ones. The output layer has 36 excitatory neurons and one inhibitory neuron. Each excitatory neuron is connected to all excitatory input neuron. As the input layer, one inhibitory neuron provides a feedback signal, which is necessary to implement the winner take all (WTA) learning scheme. Figure 1.5 shows the SNN structure.

The learning process starts with setting random weights to each synapse. After that, the patterns, which are letters from the English alphabet, are fed to the network one by one. Each pattern is a 32x32 binary matrix, every pixel is directed to one

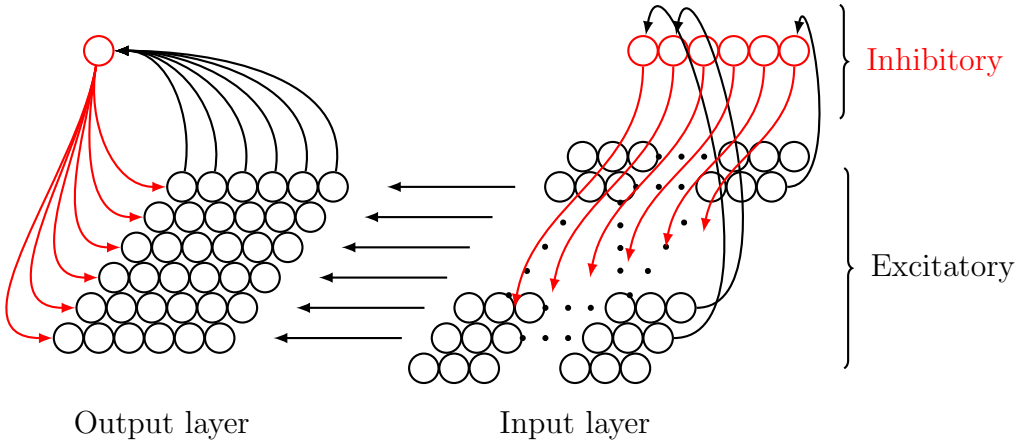


Figure 1.5: SNN for character recognition

excitatory input neuron. If the pixel is part of the letter, the corresponding neuron fires. The firing activity is then passed to the inhibitory input neurons and the excitatory output neurons. The signal at the excitatory output neuron is, simply, the dot product of two vectors, one represents the input signals coming from the input layer in the form of spikes, the other is the random synapse weight vector. The weight vector of each excitatory output neurons is called the receptive field for that neuron, which represents the input pattern that will lead to the firing of the neuron. When an excitatory neuron fires, it activates the inhibitory neuron which prevents other excitatory neurons from firing. In other words, when a neuron fires, it prohibits the others from learning the current letter, which is the implementation of the WTA method.

Simply put, learning of this particular SNN is the process where the receptive fields of the excitatory output neurons are adjusted to memorize the corresponding alphabet. The fact that an output neuron memorizes a letter implies when the letter is received at the input layer, it will lead to a strong excitation signal projected from

the input layer to this specifc neuron, which will cause the generation of a spike by such neuron.

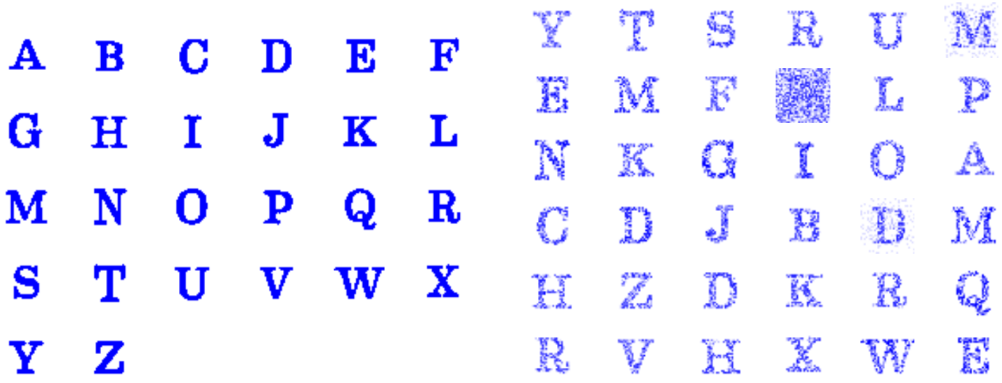


Figure 1.6: Input and output of SNN for character recognition

This process is carried out for all the letters in the alphabet sequentially. The simualtion is run for 5000 biological steps for every letter. Figure 1.6 shows the input and the output of the learning procedure.

2. METHODOLOGY

Scaling of devices comes with the price of increasing variation of key parameters that directly affects the performance of integrated circuits. The increase can be ascribed to two main factors: the inability to improve manufacturing tolerances to match the pace of the scaling, and the fact that the technology is approaching the fundamental scaling limits[12]. The variation may be so severe such that, in technologies at the 32nm range or beyond, two adjacent transistors will have different characteristics. To give an example from the recent technology data, the standard deviation of the threshold voltage (V_{th}) and mobility (μ) in the 65nm technology were 9% and 16%, respectively [18]. Therefore, it should be assumed that the fabricated circuit will significantly deviate from the designed values [1], which means that it is essential that the behavior of the design, away from the nominal values, be considered. This requires a substantial change in the design practices, moving from deterministic approaches to probabilistic and statistical ones [18, 1].

Traditionally, two methods were used to simulate the variability of circuits: process corners and Monte Carlo simulations. In process corner simulation, key device parameters are changed to obtain worst and best case performance under variation, besides the nominal values. The corners are usually provided with the technology libraries. Since it uses limited number of cases, it is time efficient, which makes attractive. However, this method is limited in its ability to account for intra-die variations. Also, it offers minimal insight to the robustness of the circuit [4]. The Monte Carlo analysis is an iterative method where random variables are used to sample the variability space. This technique gives useful and accurate information about the design, given that a large number of samples is used. Moreover, it is flexible and

can incorporate intra-die variability. The major drawback of this method is that it requires large number of samples. This translates into a computational complexity for modern integrated circuits, where millions of devices are integrated into a chip.

Another popular method for digital circuits analysis is statistical timing analysis (STA). STA is used to validate the timing performance of a digital design, in contrast to functionality, by computing the worst case delay of a circuit by propagating the worst case arrival time at the nodes of a circuit [3]. It calculates the delay at the output of a gate by adding the delay of the gate and the arrival time of the inputs, then assigns the maximum of these values to the gate. As the name implies, it is a static technique which is independent of the input vector. The accuracy, computational efficiency, and reliability of STA have made it the default choice. Conventional STA analyzes the circuit at the worst (and best) corners, which leads to the same issues with corner analysis. It tends to be pessimistic and estimates delays that rarely occur. Furthermore, like the corner analysis, STA does not provide a method of estimating yield. Several modifications have been proposed to address these issues. Specifically, methods based on probabilistic models of variations, which are collectively known as statistical STA (SSTA). These analyses deal with the delay as random variables rather than fixed numbers [12].

Previous work in the analysis of neuromorphic computing robustness has been primarily done at the circuit level, without examining the overall application performance of an ANN. In [13], the authors analyze the effect of the variation of V_{TH} on the performance of the neuron circuit. They ran Monte Carlo simulations at different standard deviation ($\sigma_{V_{TH}}$) values to obtain the mean and standard deviation of the spike frequency to determine the critical transistors of the design. In [16], three neuromorphic circuits were designed and implemented. The effect of the device mismatch on both the neuron and synapse circuits is studied. Monte Carlo analysis

is used to determine the spiking activity and synaptic weight changes due to mismatch. While in [7], simulated a feedforward ANN that implements logic functions. The authors test behavior of the circuit under stuck-at faults. In order to obtain the probability of correct operation with respect to the number of faulty devices, Monte Carlo simulations is performed while capturing process parameters variations. In the works discussed so far, all the analysis is done on the circuit level without assessing the effect the circuit faults might have on the application.

In [15], the author takes a different approach to model the circuit faults. Faults are described in terms of the change in the logical functionality of the circuit. Also, it is shown that traditional stuck-at faults, which is a gate level model, can exhibit a significantly different behavior than transistor-level hardware faults. Therefore, the author argues that a transistor level fault model should be used. Then, the paper proceeds to describe the simulation of the ANN after adjusting its logical functions according to the injected faults. The ANN is re-trained and the accuracy is determined with respect to fault-free learning. This research utilize an abstract way to express the circuit faults. Rather than considering the change of the performance due to PVT variations, it only considers faults as shorts or opens. In [2], the authors investigate the effect of synapse weight errors in the behavior of an object recognition ANN. Three types of errors are studied. The first is noisy synapses. The second is turning off a portion of the synapses, i.e. setting the weight to zero. The third is assigning the maximum allowable value to part of the synapses. The previous two works focus on the application level of the problem without considering the underlying circuit.

In this thesis we introduce a method to quantify the effect of PVT variations, along with input vectors, on the performance of the whole system at the application level. The research consists of three stages: variation analysis of the neuromorphic

circuit, behavioral level application analysis, and projection of the variation results into the application level. Variation analysis using Monte Carlo simulations is applied to the circuit to assess its performance. The accuracy of the results along with the simplicity and ease of implementation makes this method appealing. In addition, a dynamic way of simulation is required since the impact of the input vectors on the performance is part of the study. Considering the fact that the circuit under investigation can be reduced to a relatively small size, which will be be discussed in the next section, the computational cost of Monte Carlo simulation is manageable. Timing errors will be the measure of success or failure at this level.

After that, the learning function of an ANN, which performs unsupervised character recognition, is tested in the presence of faults. Namely, two types of faults are examined in this scope: integration and learning failures. A parameter is defined to reflect the ability of the ANN to learn all members of the alphabet. Finally, the results of the circuit simulation will be mapped to the application level to determine the impact of the variations on the application.

2.1 Neuromorphic Circuit Analysis

The performance of the circuit introduced in chapter one will be investigated under process, voltage, and temperature (PVT) variations. Specifically, the neuron and learning elements are targeted as these elements are the parts of the design that mimic the biological behavior. While the memristor array represents the synapse connection, it is still an evolving technology, and models and libraries describing its behavior at the circuit level are not readily available. Therefore, the memristive array is excluded from the analysis. In terms of process parameters, threshold voltage (V_{TH}) and device dimensions, namely, channel length (L), channel width (W), and oxide thickness (T_{OX}), will be subject to variations. Moreover, since the device

parameter variation is the dominant factor in delay variability [12], the interconnect delay and parasitics are not included in this research, i.e. the interconnect is assumed ideal.

The analysis of the circuit is carried out in three steps: first, preparing the design for simulation, running the simulation, and error analysis.

2.1.1 Netlist Preparation

The design is provided in the Register-transfer level (RTL), namely, as a Verilog code. However, a transistor level description is needed for PVT variation analysis. Thus, the design will be synthesized to obtain the low level circuit description. The tool used for this purpose is Design Compiler which is a Linux-based logic synthesis tool from . In addition, a commercial 90nm CMOS technology is used.

The result of the synthesis is a logic gate netlist in Verilog code, which is one level away from the target. Hence, we use v2s to perform Verilog-to-SPICE conversion. At this point, the design is described at a transistor level in a netlist suitable for simulation. This process is illustrated in 2.1.

As mentioned in the introduction, circuits can no longer be treated as a uniform unit due to PVT variation. Variations can be classified into two categories: intra-die, or local, variation and inter-die, or global, variation. Inter-die variations are the variations between the same device on different chips, i.e. the same shift occurs to all the devices on the same die. In contrast, intra-die variations are the variations between devices on the same chip, i.e. a device parameter varies between different locations in the same die. Adding the global variation can be applied directly to the netlist of the synthesized design. However, the local variations require modification of the netlist, which will be the last part of the design preparation for simulation.

Each circuit will be split into 16 local regions of similar size in terms of the

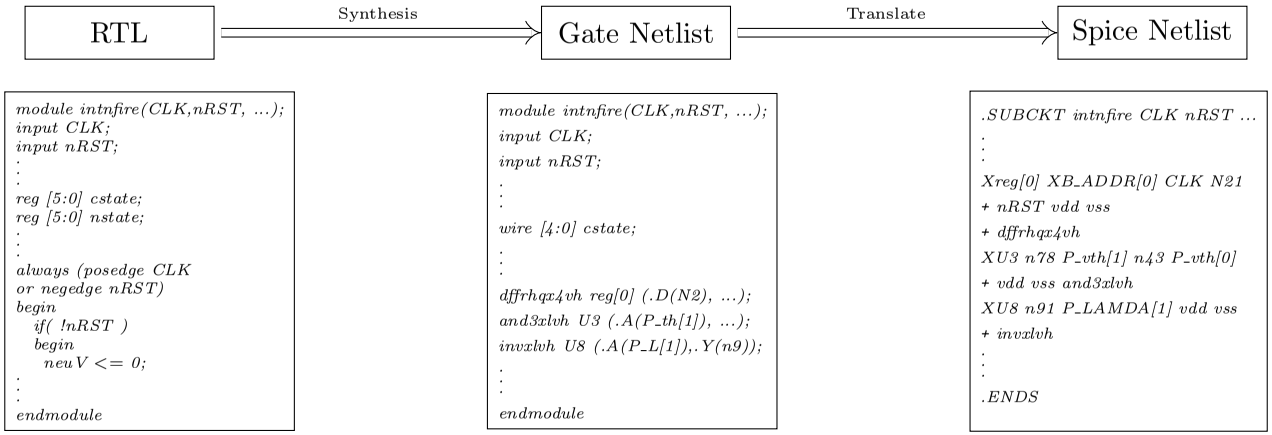


Figure 2.1: Synthesis process

number of transistors. This is an empirical approach to model the actual phenomenon observed in the fabricated chips. The division is based on the connectivity of the gates meaning that two gates which are connected will be assumed to reside next to each other. In addition, each gate will be in the same local region, i.e. the transistors of the same gate will not be on different local regions. A script is written in Python language to perform a search similar to breadth-first search (BFS) to traverse the netlist while tracking the count of transistors. The gates and the wires linking them can be considered as the nodes and edges, respectively. In this search, a gate is selected as a starting point, then its neighbors are considered, and then the neighbors of the neighbors, until the required transistor count is reached. Then the process is repeated till the whole circuit is split into the desired number of regions. Figure 2.2 shows the division algorithm.

2.1.2 Running the Simulation

The next step is performing the simulation. At this step, Monte Carlo analysis is performed on the learning and neuron elements. The tool that is used to perform the transistor-level simulation is HSPICE. Moreover, the simulation is based on a standard model library of a commercial 90nm CMOS technology which uses the BSIM4 model.

Each process parameter's nominal value in the BSIM model is replaced with a new random one. This value is composed of three components, as in equation 2.1:

$$P_{tot} = P_{nom} + \Delta P_{inter} + \Delta P_{intra} \tag{2.1}$$

where P_{nom} is the nominal value of the parameter, ΔP_{inter} is the inter-die, or global, variation, and ΔP_{intra} is the intra-die, or local, variation. Since only global variation is assumed for environmental variables, the last term in equation 2.1 is equal to zero

```
1  gates = all gates in Spice file
2  while gates not empty do
3      while count ≤ target do
4          local += gates[0]
5          gates -= gates[0]
6          count += no. of Qs in gate[0]
7          conn_g = gates connected gates[0]
8          for gate in conn_g do
9              local += gate
10             count += no. of Qs in gate
11             gates -= gate
12             if count ≥ target then
13                 break
14             end
15         end
16     end
17 end
```

Figure 2.2: Division into local regions algorithm

Table 2.1: BSIM parameters

Parameter	**BSIM Variable**
V_{TH}	`vth0`
L	`L`
W	`W`
T_{OX}	`toxp`

for these variables. Table 2.1 lists the process parameters and their corresponding variables in the BSIM model [17].

In order to implement the local variation, a new model card is added for the devices in each local region. For instance, the PMOS model card has 16 replicas, all the same except for the variation parameters. The difference is that the PMOS device and the variation parameters are indexed to allow access to each regions' specific parameters.

Although HSPICE is equipped a Monte Carlo analysis tool, an independent script is developed to generate the random variation to allow more control over the process. A Python script generates a random set of data based on the required distribution then modifies the netlist and the library accordingly.

For each global point, a number of local points and an input vector is generated. As such, each global point represents one fabricated chip with as many neurons as the instances of local points. Also, since the learning and neuron elements are on the same chip, the same global variation is applied to both. A Gaussian distribution is assumed for the random variables. Further, inter-die and intra-die variation are considered to have identical distributions. Additionally, all variables are treated as independent of each other. After generating the required variation and the input vector, the simulation is run. The algorithm in Figure 2.3 (a) depicts this process.

Please note that in this algorithm, the local point includes the input vector along with PVT variations components.

2.1.3 Error Analysis

The final step in the simulation of the circuit is the error analysis. Namely, timing errors are investigated only in this research. To determine if an error occurred or not, each chip is simulated with the same inputs under nominal values and the results are compared. In other words, the circuit is simulated for the same input twice, one with variations and the other without any variation. After that, the output of each register in the design is measured at each falling edge of the clock, which is done because the flip flops are rising edge activated. Then a comparison is performed between the values of the typical run and the run with variations. If the logical values of one register, or more, of the circuit do not match, the current realization of the circuit is considered faulty. This is a binary decision where the number of violations is not considered. An algorithmic representation of the process is presented in Figure 2.3 (b). Therefore, the outcome of the analysis at this level is the failure rates of LEs (FR_{LE}) and NEs (FR_{NE}) for each chip. Using a percentage, i.e. the rate, provides a more general representation, independent of the network structure.

2.2 Behavioral Level Analysis

As explained in the introduction chapter, there are two types of neurons: excitatory and inhibitory. In the neural network under investigation, there are 1060 excitatory and 6 inhibitory. The inhibitory output neuron is clearly essential to the application since it is responsible for implementing the winner-take-all (WTA) method. Hence, it will be assumed that this specific neuron is designed to be robust and immune to PVT variation. The same will be applied to the other inhibitory neurons as the simulations, which will be presented in the following chapter, proved

```
generate n global points
for g_point in global points do
    generate m local points
    for point in local point do
        Spice simulation with g_point,point
        Spice simulation under typical conditions
        Check for errors
    end
end
```

(a)

```
for each clk neg. edge do
    for each DFF do
        if Q_typical == Q_variation then
            Pass
        else
            Error
        end
    end
end
```

(b)

Figure 2.3: Algorithm of (a) simulation and (b) error checking

Table 2.2: Neuron types and functions

Neuron Type	**Integrate**	**Learning**
Inhibitory-Input	✓	✗
Inhibitory-Output	✓	✗
Excitatory-Input	✓	✗
Excitatory-Output	✓	✓

them to be essential as well. Therefore, the analysis will focus on the excitatory neurons.

Errors in a neuron can occur at two points: integration and learning. An integration fault happens when the calculation of the total input to the neuron is not correct. In other words, any problem occurs in the process described in equations 1.1 and 1.2. A learning error results if a neuron fails to update the weights of the synapses attached to it. That is, a flaw exists in the mechanism characterized in equations 1.3 through 1.5. Since the input layer responds to the incoming pattern and there are no learning involved in its operation, it can experience the integration error only. Conversely, the output neurons do exercise learning activity and may encounter both errors. Table 2.2 Summarizes the functions each type of neuron performs, which may experience an error.

2.2.1 Error Simulation

To simulate these errors, we turn off the corresponding function of the neuron. That is to say, a faulty neuron is considered to have no output rather than having a noisy one. This means that a failure in the integration function of the neuron will result in a silent one, while a failure in the learning function will not prevent the neuron from firing. The number of possible error combinations is huge in the ANN under study, hence simulating all of them is computationally prohibitive. Alterna-

tively, a subset of the combinations will be exercised, while the response of the ANN at other points will be extrapolated based on these.

To inject faults to the ANN, a percentage of its neurons are selected randomly, then their corresponding functions are turned off. After that, the simulation is carried out under such configuration. The output layer has a large number of neurons which allow more freedom in sampling, hence a percentage of failing neurons is used; however, the output layer has a low number of neurons which limits the choice. Consequently, the percentages permitted by the output layer will dictate the process.

One issue rises in this setup. The neurons for the input and output layer reside on the same chip and are identical. Therefore, they are subject to PVT variations in the same way. To model this, there is no differentiation between those two layers when applying the variations during the circuit simulations. Moreover, the neurons that fail to integrate are not assigned to one specific layer, rather, all possibilities are considered. This means that for a given number of failing neurons, the simulation is carried out for the cases of all neurons in the output layer, all in the input layer, and any combination between these extreme cases.

Figure 2.4 shows a possible combination of points for simulation. The x-coordinate is the number of neurons with integration error and the y-coordinate is the number of neurons with learning error. The points are chosen such that they compose a uniform grid covering possible errors from the circuit level. In addition, this grid will be refined where the circuit data points are concentrated. This is just an illustration, the actual graph is four dimensional due to the issue discussed earlier. The four dimensions are: learning, input integration, output integration failures, and the error.

In addition, the injection of the two types of faults during the simulation is carried out independently. That is to say, when sampling the population of the neurons is

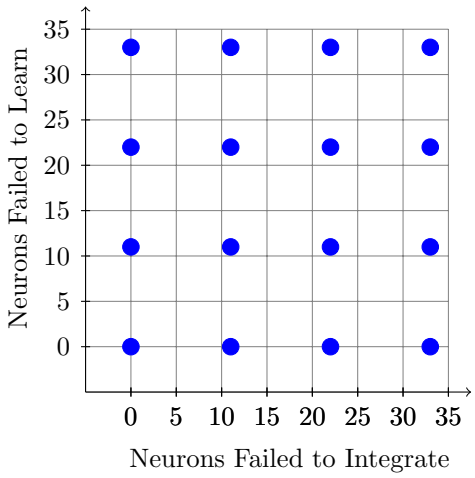


Figure 2.4: Simulation points of ANN

done, neurons to be failed in integration or learning are selected separately and independently. However, this does not mean that they are mutually exclusive. In fact, since the learning depends on the firing activity of the neuron, which in turn depends on the integration process, when a neuron fails to integrate, it, naturally, fails to learn, too.

2.2.2 Error Analysis

After the learning process is concluded, the receptive fields of the excitatory output neurons are inspected to determine the total number of letters learned. An error in this scope is defined as the ANN failing to learn one or more letters. Unlike the circuit, the failure measure is not binary at this level, i.e. either the ANN fails or succeeds. Rather, a failure percentage is calculated to express the result of the run. The term Miss-classification rate (MCR) is used to express this quantity. It is defined as follows:

$$Miss\text{-}Classification\ Rate\ (MCR) = \frac{No.\ of\ missing\ letters}{No.\ of\ letters\ in\ the\ alphabet} \tag{2.2}$$

Moreover, each case is simulated multiple times to get more accurate results. The MCR of all the runs for the same case is averaged to obtain the net MCR, as in equation 2.3.

$$MCR_{net} = \frac{\sum MCR}{No.\ of\ runs} \tag{2.3}$$

The outcome of the behavioral analysis is a 3D lookup table of MCR values parameterized in the numbers of input layer neurons failing integration, output layer neurons failing integration, and output layer neurons failing learning, respectively.

2.3 Mapping the Circuit to the Application

The last step in this research is linking the results of the circuit level and application level simulations. The robustness of the overall system is determined at this point, i.e. the impact of PVT variation on the overall performance is assessed. The results of the circuit simulations are mapped to the MCR graph created in the behavioral analysis. Then, the MCR resulting from the device and environmental variations can be calculated.

As explained earlier, the PVT variations can be classified into local and global. For each global point, many instances of local variation is simulated. As such, each global point represents one fabricated chip with as many neurons as the instances of local runs. Thus, every chip could be characterized and its MCR can be determined. To do this, three numbers should be provided for each chip: no. of output neurons failing to learn (N_{learn}), no. of output neurons failing to integrate ($N_{integrate,out}$), and no. of input neurons failing to integrate ($N_{integrate,in}$). These numbers are obtained by translating the percentages obtained at the circuit level, i.e. FR_{LE} and FR_{NE}, to the corresponding number of neurons at the behavioral level.

After that, these values can be projected on the MCR graph which is obtained in the behavioral analysis. However, there is a limited number of points in this graph,

which do not cover the whole space. Accordingly, an interpolation technique need to be applied to obtain the MCR value for the points projected from the circuit simulations. Since this is a four dimensional space, a multivariate interpolation approach is required. The trilinear interpolation method is used to this end, which is an extension of linear interpolation. Essentially, it is a weighted sum of the values associated with the points around the point in question. It can be though of as a linear interpolation in one dimension first, then in another, and so on. A full description of the trilinear method is provided in appendix A.

As discussed earlier, the circuit implementation does not distinguish between the neurons in different layers. Hence, the neurons that fail to integrate can be located at any layer. To address this, for each case, all possible combinations, of output and input neurons, are mapped to the application layer and a weighted average of their corresponding MCRs is assigned to that case. In other words, one data point at the circuit level is expanded to multiple points to account for all conceivable scenarios. Then, the MCR resulting from each scenario is estimated. However, these scenarios are not equally likely to happen. Therefore, the total MCR is the sum of each combinations' MCR weighted by its probability. The probability distribution in this case is hypergeometric. Equation 2.4 shows how the MCR for a point from the circuit level simulation is calculated.

$$MCR(FR_{NE}, FR_{LE}) = \sum_{i=1}^{m} Prob(N^i_{integrate,out}, N^i_{integrate,in}) \cdot MCR_i(N^i_{learn}, N^i_{integrate,out}, N^i_{integrate,in}) \tag{2.4}$$

where FR_{NE} and FR_{LE} are the failure rates for a chip obtained at the circuit level, m is the number of possible input-output combinations, $Prob(\cdot)$ is the probability of an

Table 2.3: Example of integration error cases

	Learning Failure	**Integration Failure**		Weight
	No. Output Neurons	No. Input Neurons	No. Output Neurons	Probability
case 1	3	0	3	0.007
case 2	3	1	2	0.094
case 3	3	2	1	0.391
case 4	3	3	0	0.508
			total	1.00

event, $N^i_{(.)}$ is the number of input and output neurons failing for the ith combination as defined earlier, and MCR_i is the estimated MCR for the specific combination. If the total number of neurons corresponding to the failure percentages obtained from the circuit analysis is not an integer, it is rounded to the nearest integer. For example, suppose that an ANN has 80 neurons at the input layer and 20 at the output layer. Also, assume that 3% is the failure rate for both learning and integration, i.e. 3 failing neurons in total. Table 2.3 shows the possible combinations and their weight. Equation 2.5 shows the total MCR calculations.

$$MCR = 0.007 \cdot MCR_1 + 0.094 \cdot MCR_2 + 0.391 \cdot MCR_3 + 0.508 \cdot MCR_4 \quad (2.5)$$

Figure 2.5 summarizes the methodology used in this research.

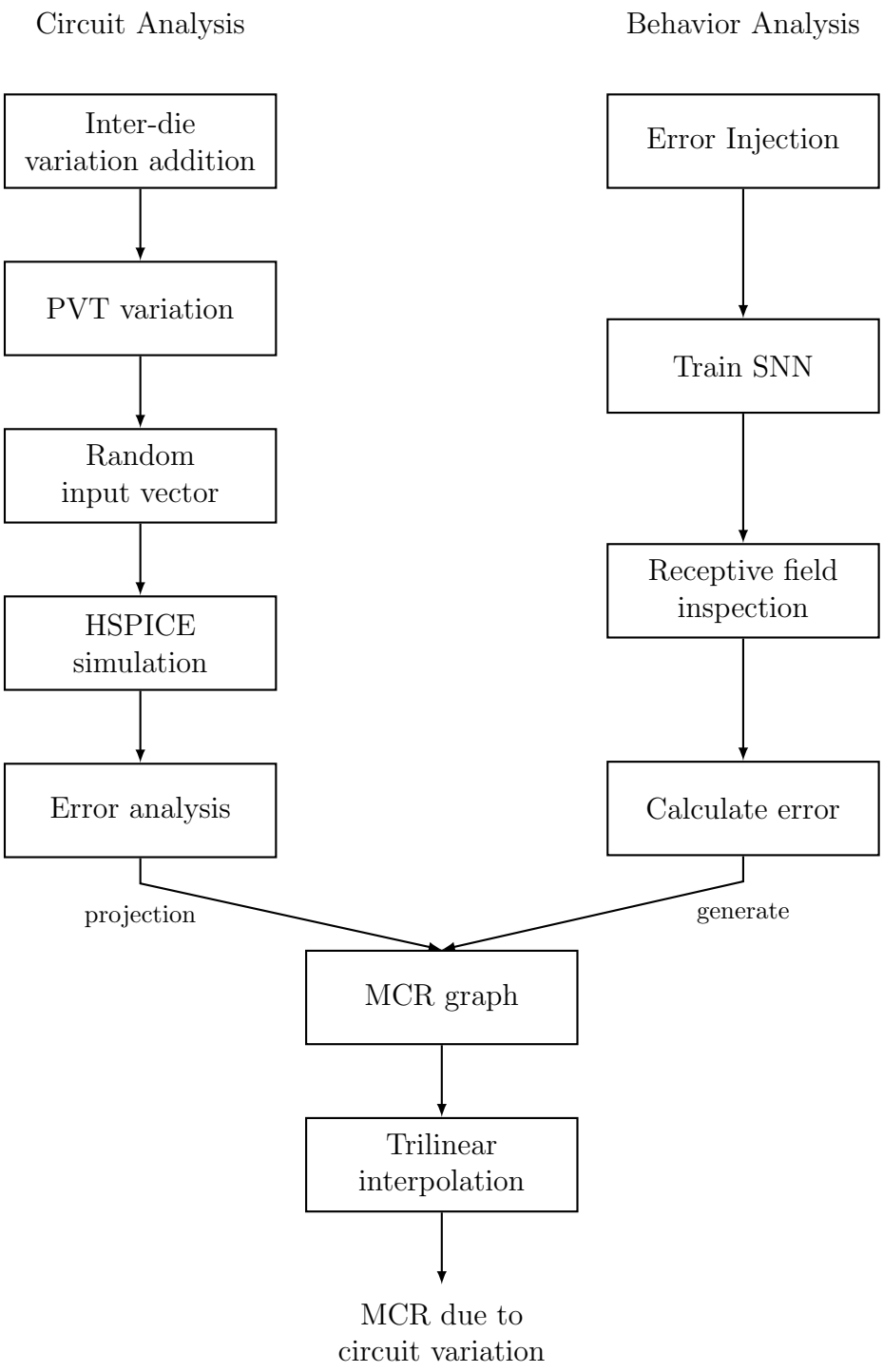


Figure 2.5: Analysis flow

3. EXPERIMENTAL RESULTS AND DISCUSSION

In this chapter the results, of the simulations of circuit and application levels are presented. In addition, the robustness of the neuromorphic circuit is discussed within the context of the method introduced by this thesis.

3.1 Neuromorphic Circuit

The input vector, process, voltage, and temperature (PVT) parameters are varied to obtain the behavior of the circuit in more realistic conditions, where the fabricated chip deviates from the designed values. The process parameters that are considered are threshold voltage (V_{TH}) and device dimensions, namely, channel length (L), channel width (W), and oxide thickness (T_{OX}). Each PVT parameter is assumed to be a random variable that has a Gaussian distribution with a mean equal to the designed specification. The clock frequency is set to 1MHz without any variation.

While the commercial 90nm library that is utilized in this study provides data for the standard deviation of the process parameters, different values are used. This is done because the research intends to cover new technologies and emerging devices, where the range of variation is larger. In these experiments, the standard deviation is set to 20% of the nominal value of each nominal parameter, which is the expected range of variation around the 40nm node [12]. To further explain, the 20% is the total variation due to global and local variations. Hence, the value of the process parameter is the sum of three elements: nominal value, random global variation, and random local variation.

The environmental variables have only one random component, namely, global. The supply voltage (V_{DD}) has a nominal value of $500mV$ and a standard deviation of $50mV$. The temperature (T) has a mean and standard deviation of 25 °C. The input

Table 3.1: Components of PVT variation

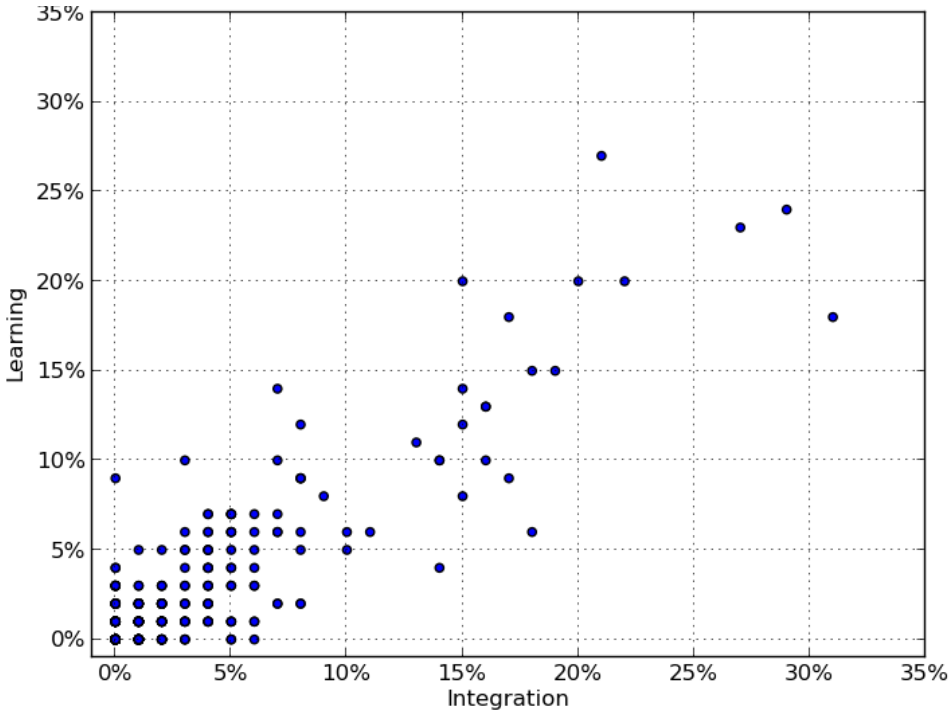
Parameter	**Global (σ)**	**Local (σ)**
V_{DD}	10%	-
T	100%	-
Process (V_{TH}, L, W, T_{OX})	10%	10%

vector is formed by randomly generating a binary vector, with a length equal to the number of inputs, then translating it to an electrical signal. Logical high is converted to the supply voltage value of the specific run. In other words, the amplitude of the logical one is assumed to have the same deviation as the supply voltage. The logical low is converted to $0V$.

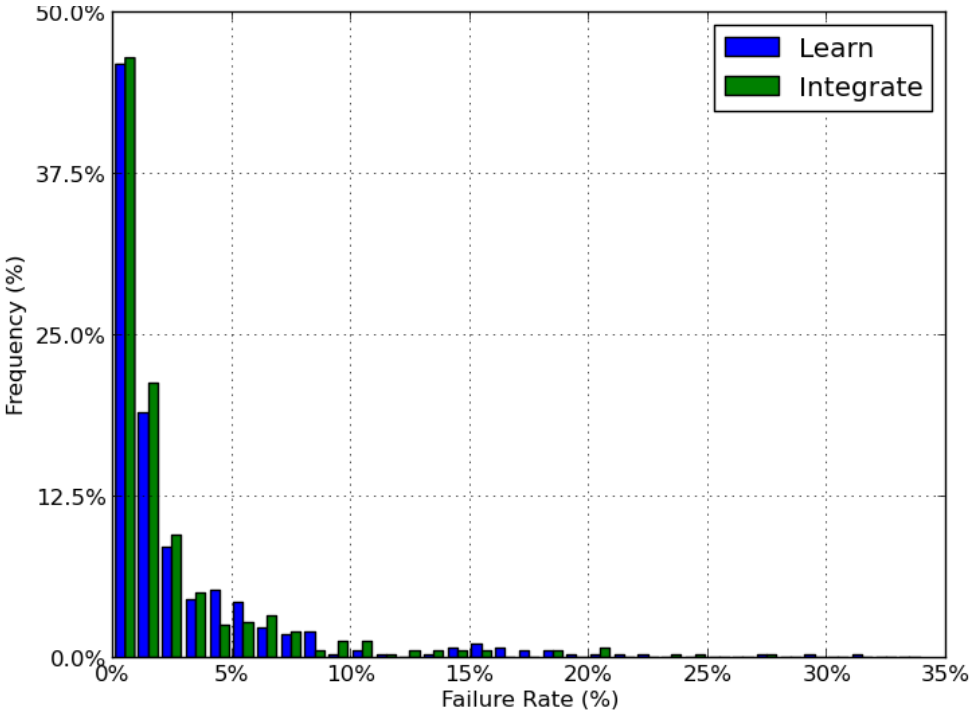
Table 3.1 summarizes the PVT parameters and their variation components as a percentage of the nominal value.

Monte Carlo simulations are carried out for 400 global points with 100 local points each, which sums up to $40,000$ different points. This configuration can be thought of as simulating 400 chips that has 100 neurons on-board. Of the 400 global runs, 139 do not exhibit an error of any kind. The worst error percentage is 31%, which happens in a chip that has 18% of the neuron elements (NE) and 31% of the learning elements (NE) failing.

In Figure 3.1a, each data point in the graph represents one chip. The coordinates are the failure rates of the neuron and learning elements. For example, a point with (1,3) coordinates means that in the simulated chip, 1% of the (NE) and 3% of the (LE) experience failure. Figure 3.1b depicts the results without the previous context. It shows how many NU and LU elements fail in total for each global simulation.



(a) Circuit failure rate



(b) Circuit failure rate histogram

Figure 3.1: Circuit variation analysis results

3.2 Application to Character Recognition

The errors that a neuron can experience are investigated. There are two stages during the operation of a spiking neural network (SNN): integration and learning. The integration operation is done by all types of neurons, hence, all neurons will be subjected to errors in integration. However, the learning process happens in the excitatory output neurons only, therefore, learning errors are injected in these types of neurons alone.

The inhibitory output neuron is clearly essential in the dynamics of the SNN. Since it is responsible for implementing the winner-take-all (WTA) method, the SNN without it is almost non-functional. If this neuron is missing, the network will learn one letter only. The same goes for the inhibitory input neuron. While they are not as crucial as the output one, the SNN is not able to memorize all the letters if there are no inhibitory neurons in the input layer. Thus, only the excitatory neurons are the focus of this study with the assumption that the inhibitory neurons are implemented with a high degree of robustness.

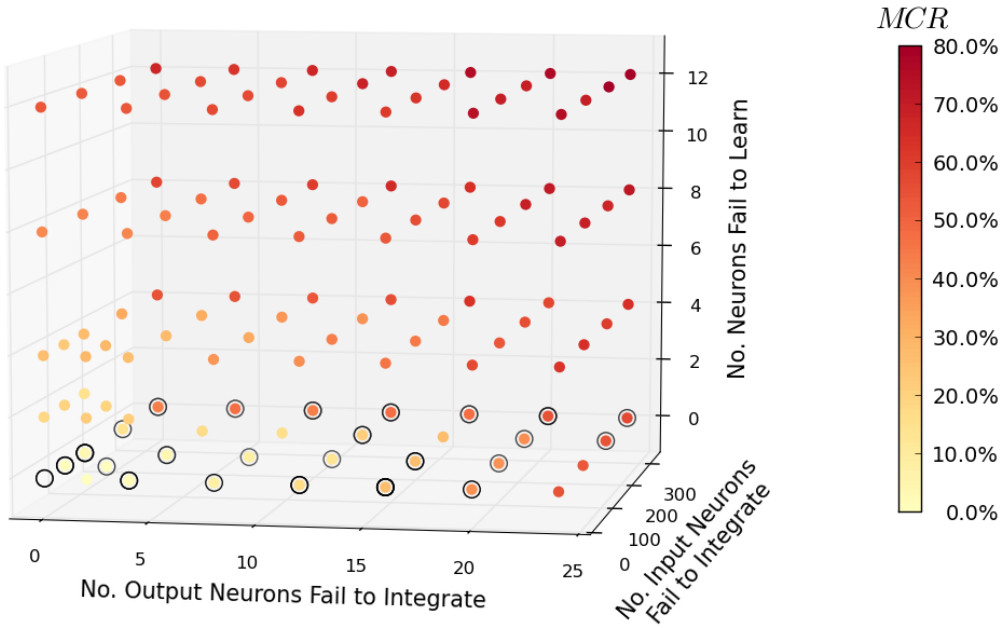
Since simulating all possible combinations is computationally prohibitive, a group of them are selected for simulation, and any other required combination are inferred from this group. The initial choice is presented in Figure 2.4. However, after the results of the circuit simulation are examined, additional set is added based on the concentration of failure points at the circuit level. Moreover, as the results shows that the SNN is more sensitive to the failures at the output layer, it is sampled more than the input layer. Also, the number of output neurons failing to integrate is capped at 24 and any point larger than that is assumed to have an MCR of 100%. This is done to reduce the computation time and justified by the fact that the probability of having larger than 24 output neurons failing is very low.

For each error injected, 20 points are simulated. Each point with a different batch of neurons failing. For instance, after deciding to insert 22.22% learning and 11.11% integration failures, these portions of the SNN neurons are randomly selected. Then, the learning process is initiated. After the learning is over, another fraction with the same number of neurons are randomly picked and the network re-learns. This procedure is repeated 20 times. Also, because neurons are chosen for integration failure independently from the ones picked for learning failure, in some runs, the same neuron might be selected to fail in both.

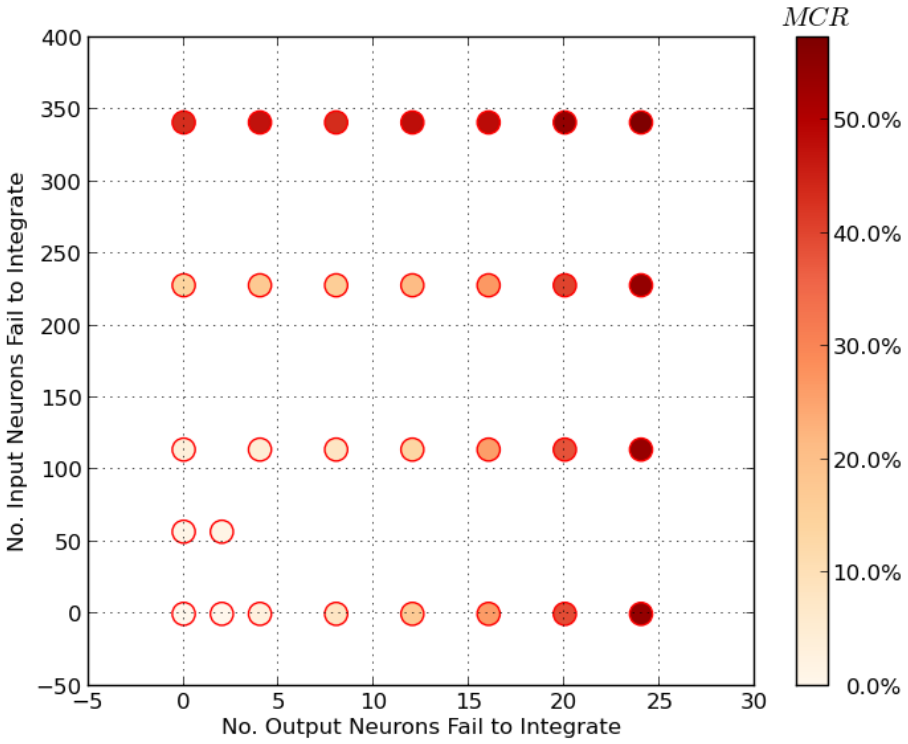
Figure 3.2a shows the performance of the SNN with error injection. The miss-classification rate (MCR), defined in equation 2.2, is color coded. The darker the point, the higher the MCR, i.e. the worse the performance. The colorbar provides the value for each degree of the red color. The results show that the network can loose up to 114 neurons while, on average, has an error of less than one letter. In some cases, all the alphabets are learned despite losing 22.22% percent of the network.

To further illustrate, Figure 3.2b shows the integration failure plane for zero learning failure. The MCR values for the points where the circuit data is concentrated are presented in Table 3.2. The first three columns are the number of neurons failing in each category. The last column is the average MCR.

Figure 3.3 compares between learning and integration failures in terms of MCR. It is clear that learning failures have a bigger effect on the performance of the SNN. The difference can be ascribed to the fact that a learning error has a higher chance of preventing the network from memorizing the input pattern. Because a learning failure does not necessarily prevent a neuron from firing, a neuron can win the competition and excite the inhibitory output neuron. By doing that, it blocks the training of the other neurons. Therefore, it does not learn the pattern, and prevents the whole network from learning it.



(a) Total response with learning and integration failures



(b) Integration only

Figure 3.2: Miss-classification rate (MCR)

Table 3.2: MCR for simulation point

Learning	Integration (output)	Integration (output)	MCR
0	2	0	0.000
0	0	57	0.962
0	2	57	1.346
0	4	0	2.692
0	0	114	3.654
0	4	114	4.615
0	8	0	8.269
0	8	114	8.269
0	0	228	14.423
2	0	114	14.615
0	12	0	17.115
0	4	228	17.500
2	0	0	17.885
2	2	57	19.423
2	0	57	20.192
2	4	114	20.385
2	4	0	21.346
2	2	0	21.923
4	0	57	23.077
2	0	228	24.038
4	4	0	25.962
4	0	0	26.154
4	0	114	27.308
4	4	114	28.269
4	2	57	28.462
2	8	0	28.462
4	2	0	28.846
4	4	228	32.308
4	0	228	32.500
4	8	114	32.885
4	8	0	37.500
4	12	0	40.192
8	0	0	40.769
8	4	0	41.538
8	0	114	42.308
8	4	114	43.269
0	0	341	43.269
8	0	228	44.423
8	4	228	46.923
8	8	114	50.000
8	8	0	50.192
8	12	0	52.500
12	0	114	52.885
12	4	0	53.462
12	0	228	53.654
12	0	0	53.846
12	4	114	54.808
4	0	341	54.808
12	8	0	57.115
12	8	114	57.115
12	4	228	57.308
8	0	341	58.269
12	12	0	63.654
12	0	341	67.692

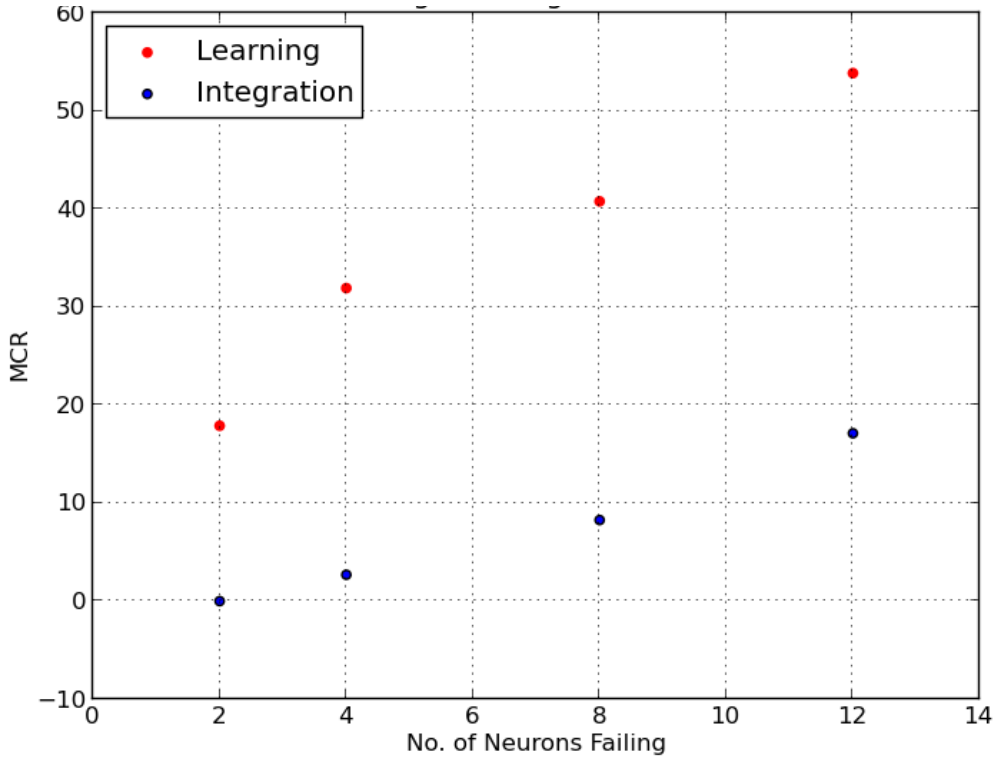


Figure 3.3: Comparison of MCR due to learning and integration failures

3.3 Overall Response of the System

In order to quantify the robustness of the neuromorphic circuit, the effect of the device variations on the application level should be determined. This is done by considering the percentages of neurons failing to integrate or learn at the circuit level. For each chip (i.e. global point), the outcome of the simulation is two values: the percentage of neurons failing to integrate and the percentage of neurons failing to learn. These two values constitute a point in the MCR graph obtained from the behavioral simulation, therefore we can estimate the MCR for the chip. However, due to the issue discussed earlier, this point is expanded to a line to consider all possibilities of integration failure. Then the MCR is determined for each point on

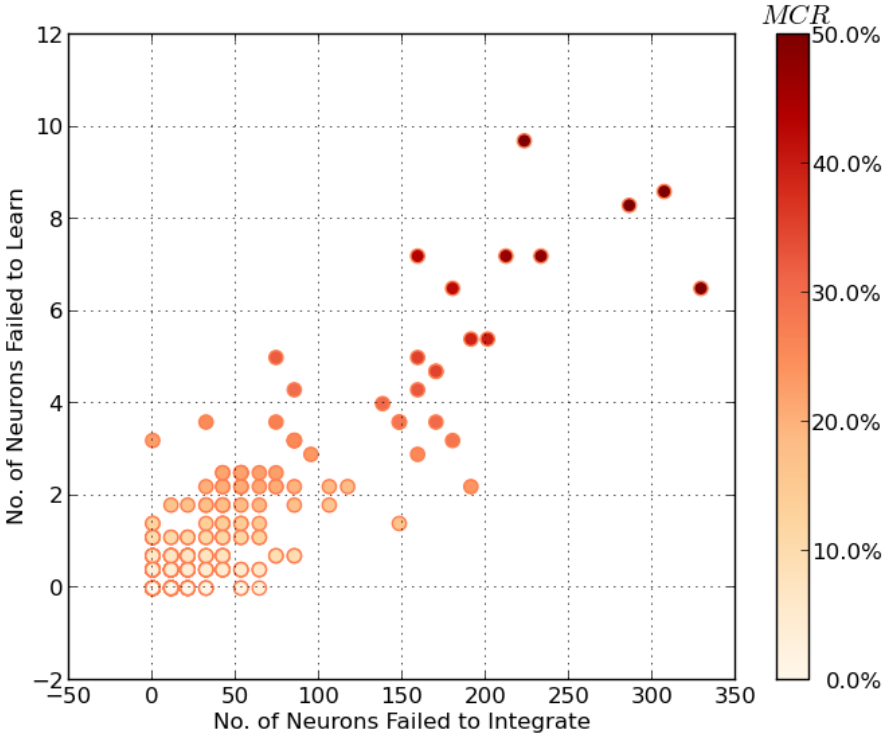
the line and a weighted sum of the MCR values are calculated. This weighted sum is the MCR value assigned to the chip.

Figure 3.4a shows the interpolated MCR for neuron failure cases obtained at the circuit level. As can be seen from Figure 3.4b, about 45% of the runs result in less than 2% MCR. In Figure 3.5, the error is expressed in terms of missing patterns. This shows that the SNN is robust against the errors propagated from the circuit level, where 55% of the cases resulted in almost perfect operation, i.e. less than one pattern missing on average. In addition, in 90% of the cases, around 5 patterns or less are not learned.

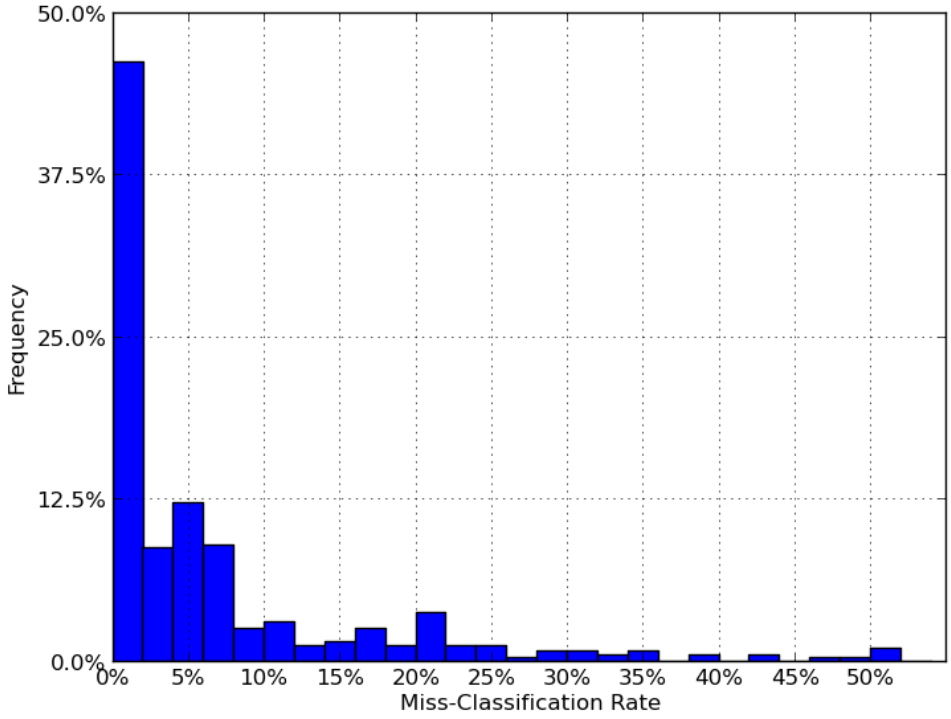
3.4 Parameter Specific Simulations

Further analysis are carried to determine the effect of key parameters on the behavior of the neuromorphic circuit. This is done to give a design perspective into the system. At the circuit level, the impact that the supply voltage has on the MCR is checked. At the application level, the number of the output neurons is varied as a way to study how the structure of an SNN influence its performance. In addition, a different pattern is used to train the network to test its efficiency in learning.

The voltage supply is fixed globally and varied locally. It is assumed to have a normal distribution with a standard deviation equal to one third of the nominal value, i.e. 3σ is equal to 10%. The temperature is kept constant at 25 °C. The process variability is retained at the same settings. The simulation is run one hundred times: 10 global points with 10 local point for each global point. The supply voltage is varied between $350mV$ and $700mV$ in $80mV$ steps. The MCR is calculated according to the flow proposed in this thesis. The resulting average MCR for each value is shown in Figure 3.6. A clear inverse relation between MCR and the supply voltage can be seen. This is expected as the higher the supply voltage, the lower the errors, and



(a) Projection of circuit results on MCR plot



(b) Histogram of MCR due to circuit errors

Figure 3.4: Mapping neuromorphic circuit variation results to the application level

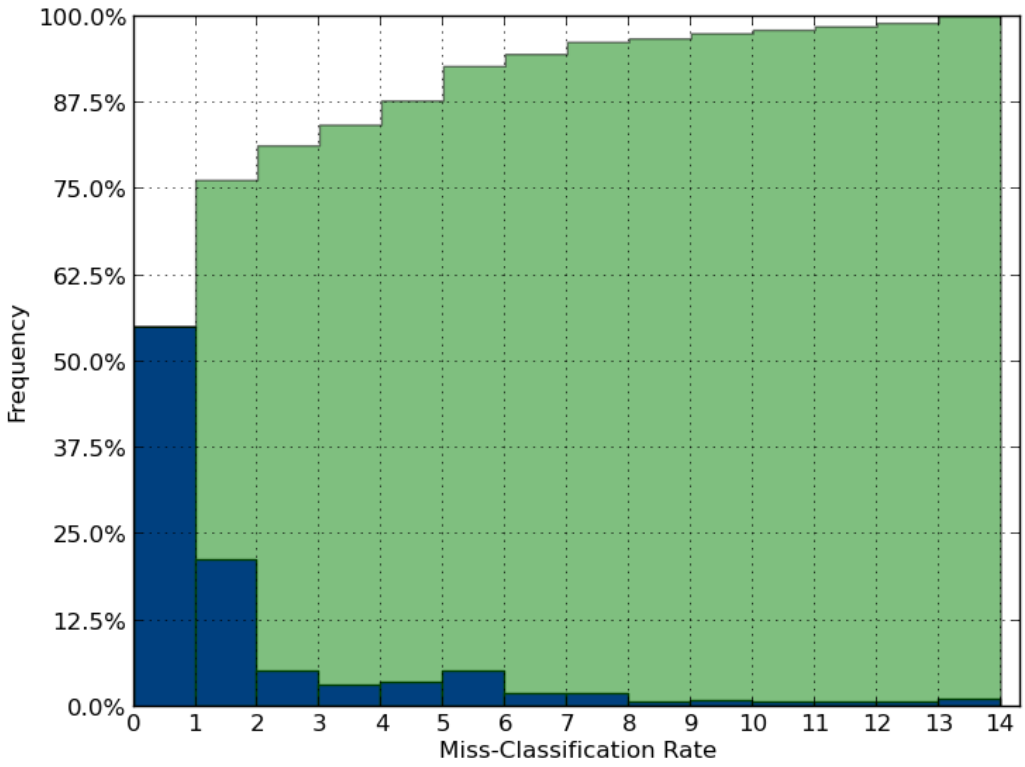


Figure 3.5: Letters missing due to circuit errors

hence the lower MCR.

Using the same methodology as in Section 3.2, the effect of the number of excitatory output neurons on the character recognition SNN is evaluated, while maintaining the same structure otherwise. Four cases are tested, 34, 35, 37, and 38 output neurons. For each case, ten runs are executed. Here, the plots for the 37 and 38 cases are presented. Data tables, similar to Table 3.2, for all of the cases and the plots of 34 and 35 cases can be found in appendix B. Figure 3.7a make a comparison between MCRs of 37 and 38 output neurons. The color of the point represent the difference calculated as follows:

$$\Delta_{MCR} = MCR_{37} - MCR_{36} \tag{3.1}$$

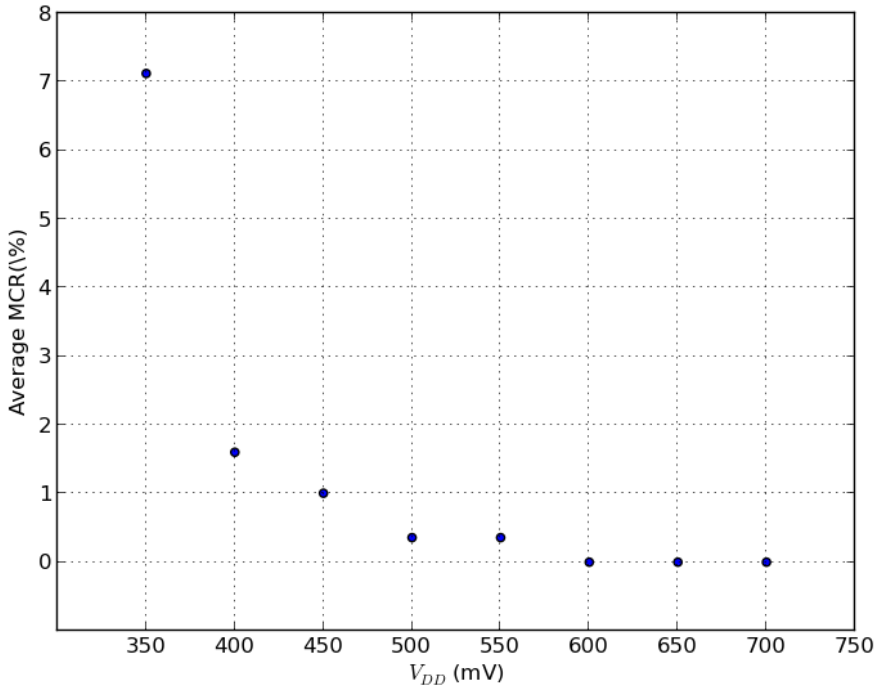


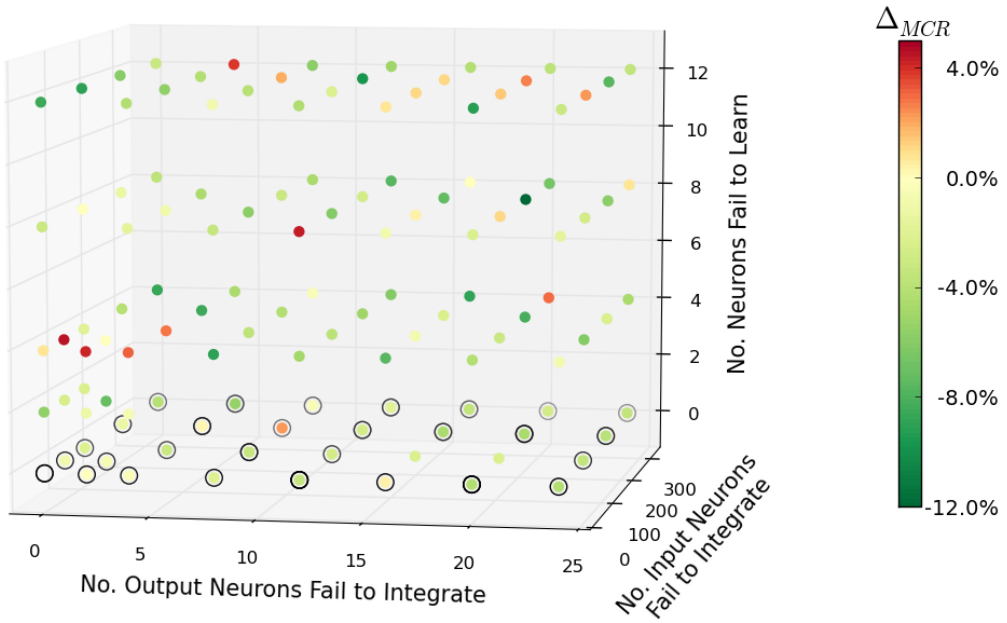
Figure 3.6: MCR due to supply voltage variability

Therefore, a minus value means that the performance of the current network is better than the original network, and a positive value means the opposite.

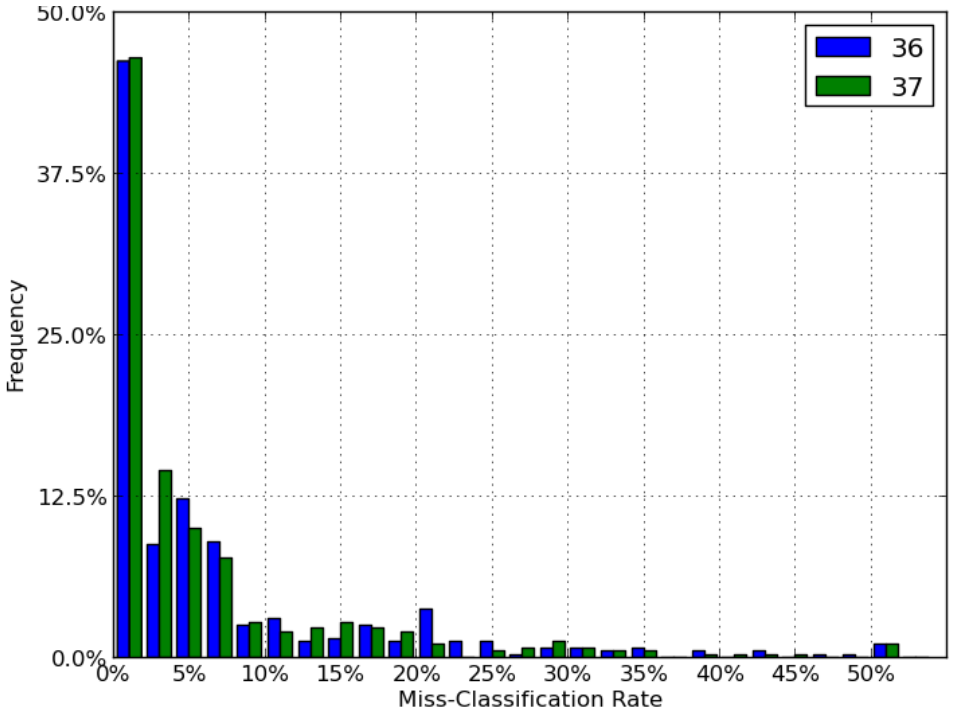
This is reflected in the graph by coloring the points according to the assumption that increasing the number of the output neurons enhances the performance. Hence a green colored point indicate that it matches the expectation, and a red one implies the reverse. In other words, for 37 and 38 output neurons a green point means negative Δ_{MCR}, while for 34 and 35 it means a positive Δ_{MCR}.

Moreover, to make a distinction based on the number of patterns, any point that has an MCR difference equivalent to more than one letter is circled.

To give a comprehensive picture, another comparison is made in terms of device variation effect. The MCR rising from errors at the circuit level is compared between the default run and the learning outcome in the 37 case in Figure 3.7b. The results

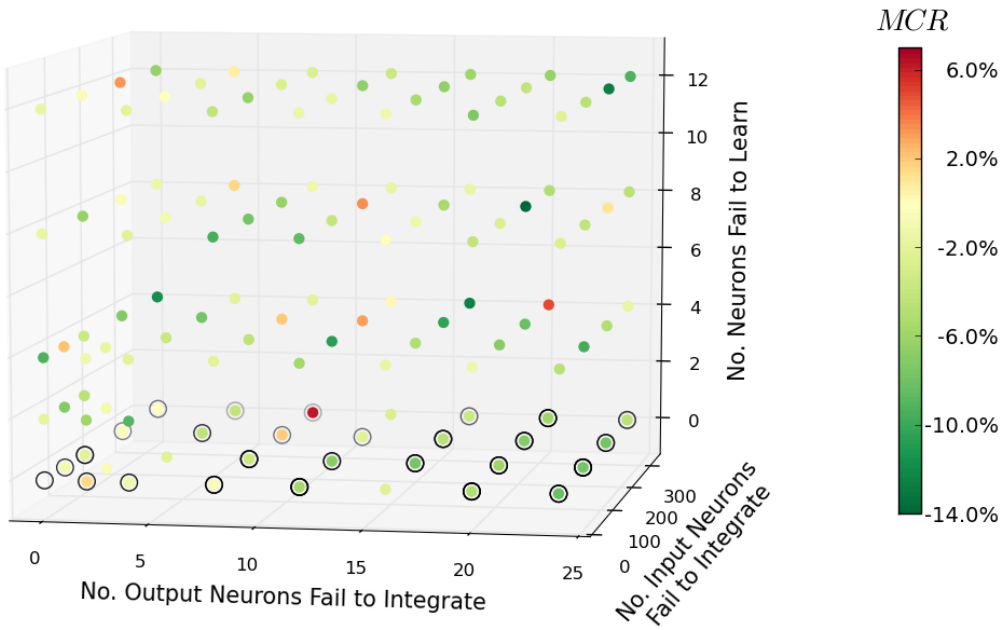


(a) 37 vs. 36 output neurons

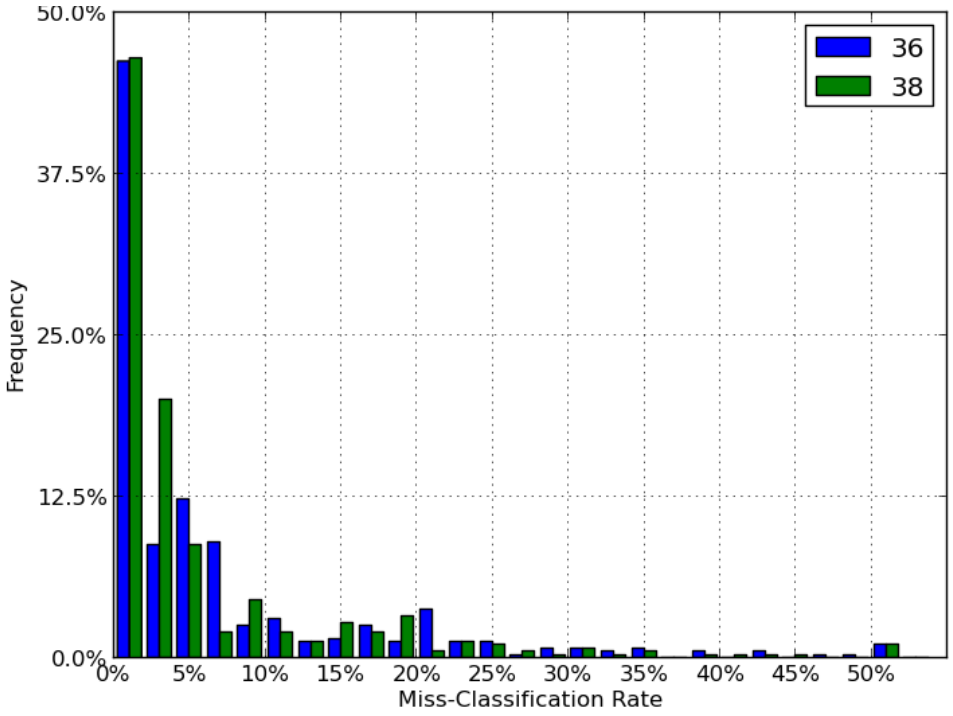


(b) 37 vs. 36 MCR due to circuit variations

Figure 3.7: MCR for 37 output neurons



(a) 38 vs. 36 output neurons



(b) 38 vs. 36 MCR due to circuit variations

Figure 3.8: MCR for 38 output neurons

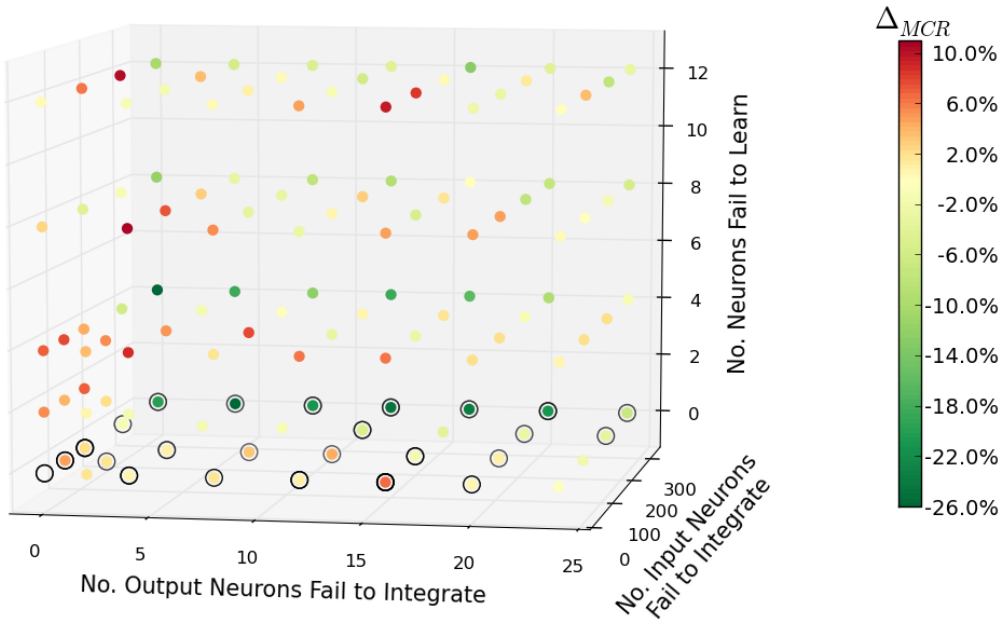
is presented in the same way for 38 output neurons in Figure 3.8a and 3.8b. It is evident that in most cases, the assumption about the number of neurons holds.

The last modification to the original run is changing the input pattern. The Greek alphabet is used to train the network instead of the English one. The alphabet is shown in Figure 3.9. The same method of testing and illustrating the results in the case of output neurons is used. The difference in MCR is defined as follows:

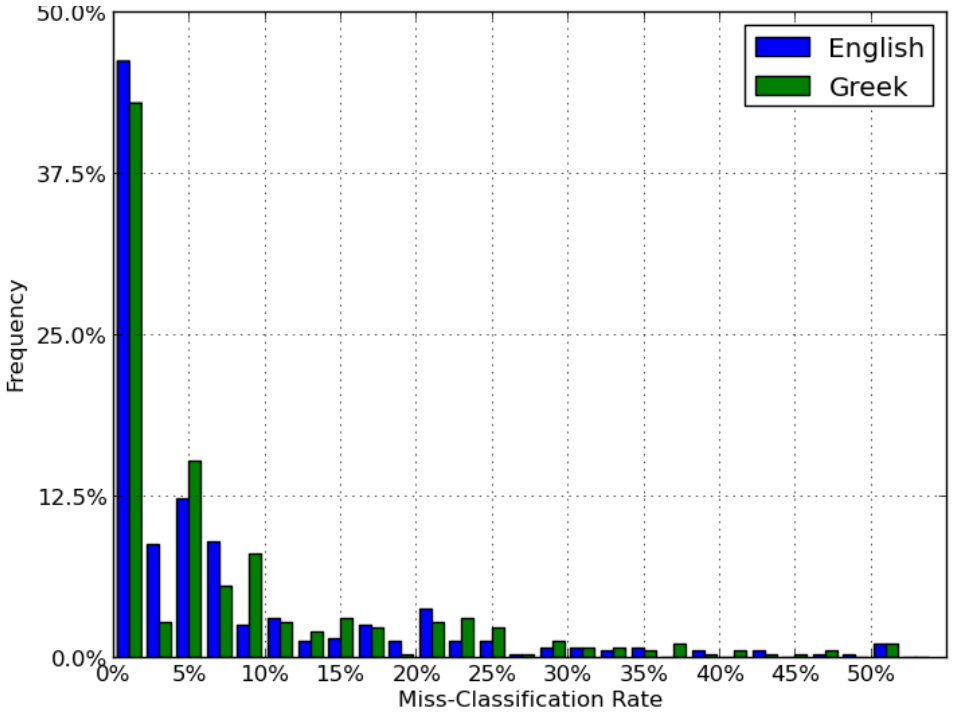
$$\Delta_{MCR} = MCR_{grk} - MCR_{eng} \tag{3.2}$$

which means that, a red point indicates a worse performance in the Greek, while a green one implies the Greek is better. Figures 3.10a and 3.10b depicts the performance of the network in learning the Greek alphabet. The efficiency of the SNN in learning the Greek alphabet is comparable to the English.

Figure 3.9: Greek alphabet set used in training the SNN



(a) Greek vs. English



(b) Greek vs. English MCR due to circuit variations

Figure 3.10: MCR for Greek alphabet

4. CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this thesis, a framework for evaluating the robustness of neuromorphic circuits in the presence of process and environmental variations is introduced. By mapping the results of circuit-level simulations to those of the application-level, the overall response of the design can be checked. Monte Carlo based analysis at the transistor-level is carried out to determine the impact that the PVT variations have on the performance of the neuromorphic circuit. In addition, a behavioral model is used to study the design's efficiency at the application level. The model is used to reduce the computational cost. The application consists of an artificial neural network that implements a character recognition function. Possible errors of such systems are identified and injected in the network to characterize the effect of faults on the system's ability to learn different patterns.

The outcome of the research indicates that the neuromorphic circuit design is resilient to errors. When the simulation is performed using a commercial 90nm technology, in 55% of the cases, the system is able to learn the English alphabet with an error of less than one letter on average. Moreover, 90% of the cases result in an error of five letters or less.

Moreover, the influence of key parameters on the system's operation is studied to provide an insight from a design perspective. On the circuit side, the supply voltage is considered. On the application side, the structure of the network and the input pattern are examined. It has been shown that raising the supply voltage drastically improves the performance. Furthermore, increasing the number of output neurons enhances the learning ability of the network. The design demonstrates a comparable

efficiency in learning the Greek alphabet to that of the English.

4.2 Future Work

This work has provided insight on the resilience of neuromorphic circuits. However, there are few key issues that need to be addressed. While the Monte Carlo analysis were feasible at this stage, it will no longer be the case as neuromorphic designs get larger and more complicated. Hence, more efficient methods are essential to carry out the analysis. Furthermore, analysis of the non-digital units within the circuit is required to have a more comprehensive view of the performance. In this research, only timing errors are considered and they were assumed to cause silent devices, which is not always the case. Therefore, other types of errors and different manifestations of these errors should be studied.

This research utilized a two-layered neural network to demonstrate the functionality of the circuit. The neural network performed character recognition. The examination of the network showed that it is robust in the existence of PVT variations. It would be interesting to investigate the performance of multi-layered network in such conditions. Moreover, further research in other applications would help elevate the confidence in the robustness of the neuromorphic circuits.

REFERENCES

[1] Shekhar Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, Nov 2005.

[2] Johannes Fieres, Karlheinz Meier, and Johannes Schemmel. A convolutional neural network tolerant of synaptic faults for low-power analog hardware. In *Artificial Neural Networks in Pattern Recognition*, pages 122–132. Springer, 2006.

[3] Rajesh Garg and Sunil P Khatri. *Analysis and design of resilient VLSI circuits*. Springer, New York, 2010.

[4] Anne Gattiker, Sani Nassif, Rashmi Dinakar, and Chris Long. Timing yield estimation from static timing analysis. In *Proceedings of IEEE International Symposium on Quality Electronic Design*, pages 437–442, 2001.

[5] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, New Jersey, 2009.

[6] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: a tutorial. *IEEE Computer*, 29(3):31–44, 1996.

[7] Neil Joye, Alexandre Schmid, Yusuf Leblebici, Tetsuya Asai, and Yoshihito Amemiya. Fault-tolerant logic gates using neuromorphic cmos circuits. In *Proceedings of IEEE Conference on Ph.D. Research in Microelectronics and Electronics Conference*, pages 249–252, 2007.

[8] Henry R Kang. *Color technology for electronic imaging devices*, volume 28. SPIE press, Bellingham, WA, 1997.

[9] Yongtae Kim. *Energy efficient and error resilient neuromorphic computing in VLSI.* Ph.d. dissertation, Texas A&M University, College Station, TX, 2013.

[10] Yongtae Kim, Yong Zhang, and Peng Li. A digital neuromorphic vlsi architecture with memristor crossbar synaptic array for machine learning. In *Proceedings of IEEE International SOC Conference (SOCC)*, pages 328–333, 2012.

[11] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Proceedings of IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, Sept 2011.

[12] Michael Orshansky, Sani Nassif, and Duane Boning. *Design for manufacturability and statistical design: a constructive approach.* Springer, New York, 2007.

[13] Sun Shuo and Arindam Basu. Analysis and reduction of mismatch in silicon neurons. In *Proceedings of IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 257–260, 2011.

[14] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, 2000.

[15] Olivier Temam. A defect-tolerant accelerator for emerging high-performance applications. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 356–367, 2012.

[16] Jayawan HB Wijekoon and Piotr Dudek. VLSI circuits implementing computational models of neocortical circuits. *Journal of Neuroscience Methods*, 210(1):93 – 109, 2012.

[17] Xuemei Xi, Mohan Dunga, Jin He, Weidong Liu, Kanyu M. Cao, Xiaodong Jin, Jeff J. Ou, Mansun Chan, Ali M. Niknejad, and Chenming Hu. BSIM 4v4.3.0 MOSFET model users' manual. Technical report, EECS Department, University of California, Berkeley, 2003.

[18] Wei Zhao, Frank Liu, Kanak Agarwal, Dhruva Acharyya, Sani R Nassif, Kevin J Nowka, and Yu Cao. Rigorous extraction of process variations for 65-nm cmos design. *IEEE Transactions on Semiconductor Manufacturing*, 22(1):196–203, 2009.

APPENDIX A

TRILINEAR INTERPOLATION

The trilinear interpolation is an extension to the 1-D linear interpolation. It is used to estimate the value at a random point enclosed in cuboid with known values at its vertices. In Figure A.1, suppose we want to find the value of point (P) based on the known values of points $(p_{000}, p_{001}, p_{010}, p_{011}, p_{100}, p_{101}, p_{110}, p_{111})$ which are $(f_{000}, f_{001}, f_{010}, f_{011}, f_{100}, f_{101}, f_{110}, f_{111})$. It can be written as [8]:

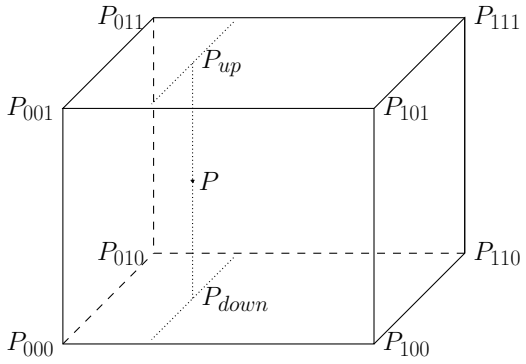


Figure A.1: Trilinear interpolation

$$f(x, y, z) = c_0 + c_1\Delta x + c_2\Delta y + c_3\Delta z + c_4\Delta x\Delta y + c_5\Delta x\Delta z + c_6\Delta y\Delta z + c_7\Delta x\Delta y\Delta z \tag{A.1}$$

where:

$$\Delta x = x - x_0$$

$$\Delta y = y - y_0$$

$$\Delta z = z - z_0$$

$$c_0 = f(p_{000})$$

$$c_1 = \frac{f(p_{100}) - f(p_{000}))}{(x_1 - x_0}$$

$$c_2 = \frac{f(p_{010}) - f(p_{000}))}{(y_1 - y_0}$$

$$c_3 = \frac{f(p_{001}) - f(p_{000}))}{(z_1 - z_0}$$

$$c_4 = \frac{f(p_{110}) - f(p_{010}) - f(p_{100}) - f(p_{000})}{(x_1 - x_0)(y_1 - y_0)}$$

$$c_5 = \frac{f(p_{101}) - f(p_{001}) - f(p_{100}) - f(p_{000})}{(x_1 - x_0)(z_1 - z_0)}$$

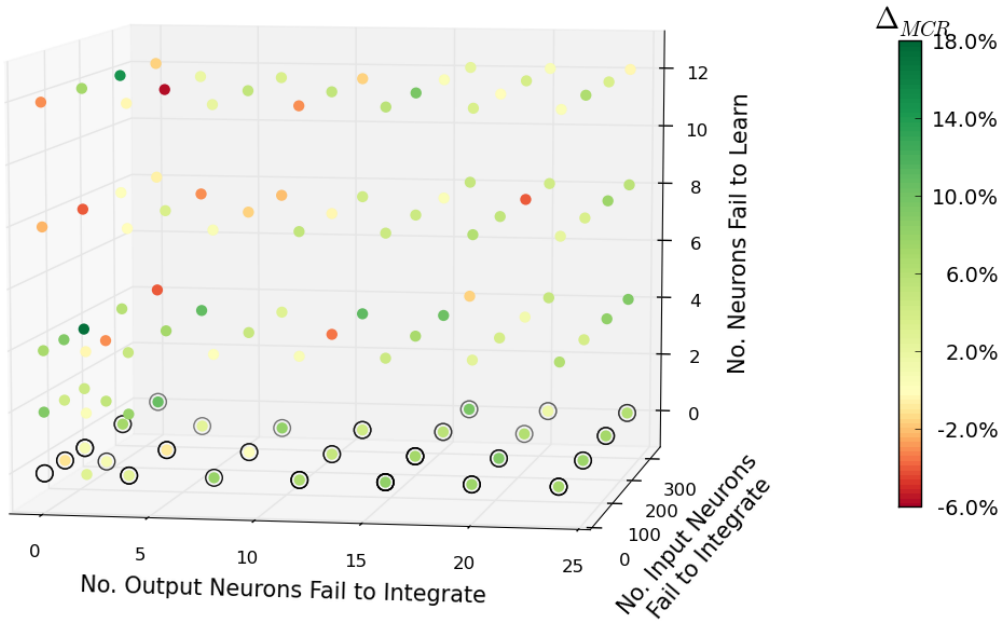
$$c_6 = \frac{f(p_{011}) - f(p_{001}) - f(p_{010}) - f(p_{000})}{(y_1 - y_0)(z_1 - z_0)}$$

$$c_7 = \frac{f(p_{111}) - f(p_{011}) - f(p_{101}) - f(p_{110}) + f(p_{100}) + f(p_{001}) + f(p_{010}) - f(p_{000})}{(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)}$$

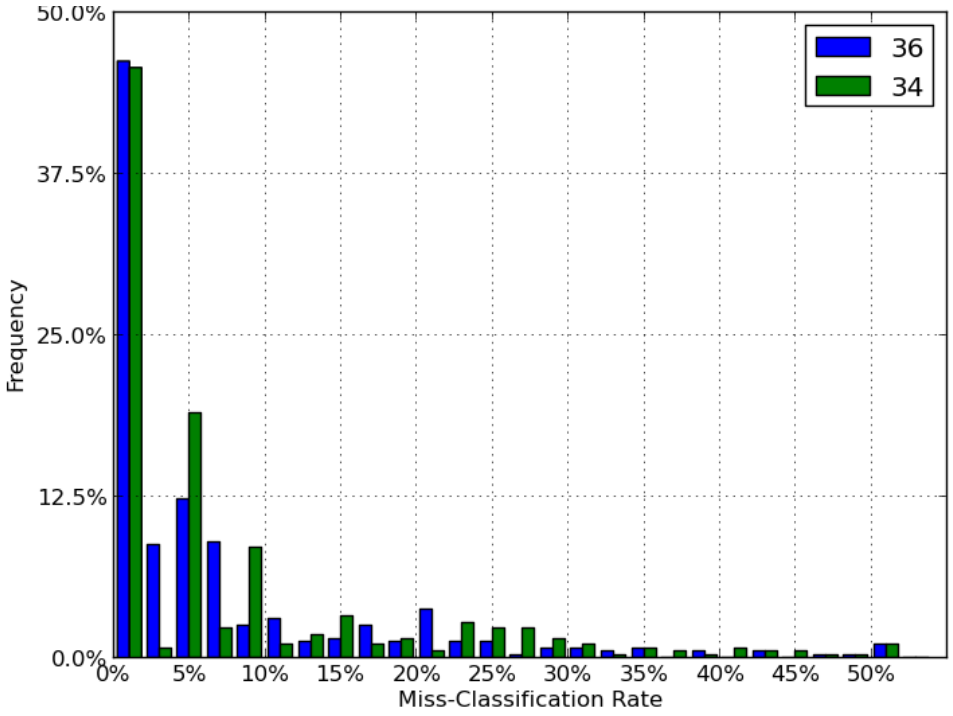
APPENDIX B

ADDITIONAL RESULTS

In this appendix, the results for the Greek alphabet and different numbers of output neurons are presented. The raw results are presented in tables B.2 and B.1, respectively. The plots for the 34 output neurons are provided in figures B.1a and B.1b, and the ones for 35 are in figures B.2a and B.2b.

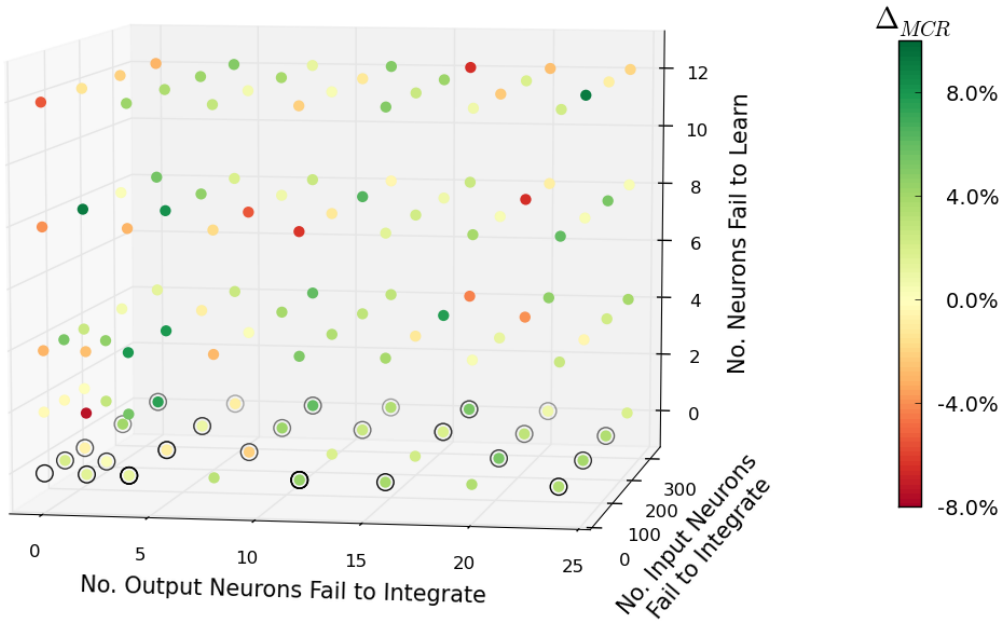


(a) 34 vs. 36 output neurons

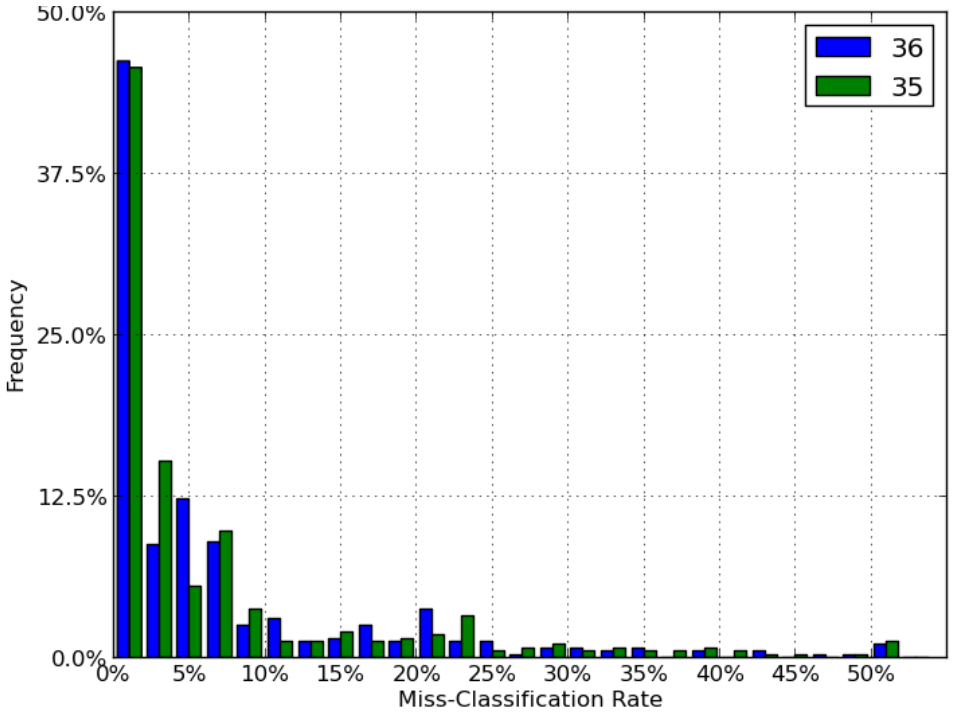


(b) 34 vs. 36 MCR due to circuit variations

Figure B.1: MCR for 34 output neurons



(a) 35 vs. 36 output neurons



(b) 35 vs. 36 MCR due to circuit variations

Figure B.2: MCR for 35 output neurons

Table B.1: Comparison of MCR for different number of output neurons

Learning	Integration (output)	Integration (input)	34	35	36	37	38
0.0	0.0	57	0.00	3.08	0.96	0.00	0.00
0.0	0.0	114	4.62	3.08	3.65	1.54	1.54
0.0	0.0	228	21.54	18.46	14.42	13.08	13.85
0.0	0.0	341	53.85	50.77	43.27	39.23	43.08
0.0	2.0	0	3.08	1.54	0.00	0.00	1.54
0.0	2.0	57	2.31	1.54	1.35	0.77	0.77
0.0	4.0	0	5.38	3.85	2.69	2.31	1.54
0.0	4.0	114	3.85	3.85	4.62	2.31	2.31
0.0	4.0	228	20.00	18.46	17.50	17.69	13.08
0.0	8.0	0	16.15	11.54	8.27	6.15	7.69
0.0	8.0	114	8.46	6.15	8.27	3.85	4.62
0.0	12.0	0	23.85	21.54	17.12	13.85	11.54
2.0	0.0	0	26.92	17.69	17.88	12.31	16.15
2.0	0.0	57	24.62	20.00	20.19	17.69	13.08
2.0	0.0	114	19.23	14.62	14.62	12.31	10.00
2.0	2.0	0	22.31	14.62	21.92	20.77	16.15
2.0	2.0	57	24.62	22.31	19.42	12.31	18.46
2.0	4.0	0	29.23	26.92	21.35	20.77	12.31
4.0	0.0	0	33.08	23.08	31.92	26.92	16.92
4.0	0.0	57	32.31	28.46	23.08	27.69	25.38
4.0	0.0	114	44.62	30.00	27.31	25.38	23.85
4.0	0.0	228	38.46	33.08	32.50	28.46	25.38
4.0	0.0	341	50.77	56.15	54.81	46.15	43.08
4.0	2.0	0	28.46	26.15	28.85	33.08	27.69
4.0	2.0	57	25.38	33.08	28.46	28.46	26.92
4.0	4.0	0	30.77	33.85	25.96	29.23	23.85
4.0	4.0	114	35.38	36.15	28.27	26.15	24.62
4.0	4.0	228	43.08	31.54	32.31	23.85	24.62
4.0	8.0	0	37.69	34.62	37.50	28.46	35.38
4.0	8.0	114	37.69	33.08	32.88	28.46	28.46
4.0	12.0	0	40.77	45.38	40.19	35.38	34.62
8.0	0.0	0	38.46	36.92	40.77	37.69	39.23
8.0	0.0	114	38.46	51.54	42.31	42.31	36.15
8.0	0.0	228	44.62	44.62	44.42	43.08	43.85
8.0	0.0	341	57.69	63.85	58.27	54.62	56.92
8.0	4.0	0	41.54	38.46	41.54	40.00	39.23
8.0	4.0	114	46.92	51.54	43.27	40.00	42.31
8.0	4.0	228	43.85	51.54	46.92	42.31	45.38
8.0	8.0	0	50.77	48.46	50.19	46.92	40.77
8.0	8.0	114	48.46	44.62	50.00	44.62	42.31
8.0	12.0	0	57.69	46.15	52.50	56.92	43.85
12.0	0.0	0	50.77	48.46	53.85	45.38	52.31
12.0	0.0	114	60.00	51.54	52.88	43.85	52.31
12.0	0.0	228	68.46	51.54	53.65	47.69	56.92
12.0	0.0	341	66.15	64.62	67.69	64.62	61.54
12.0	4.0	0	53.08	57.69	53.46	49.23	51.54
12.0	4.0	114	49.23	58.46	54.81	48.46	54.62
12.0	4.0	228	59.23	61.54	57.31	53.08	55.38
12.0	8.0	0	59.23	60.00	57.12	56.15	53.08
12.0	8.0	114	62.31	57.69	57.12	53.08	50.77
12.0	12.0	0	60.77	61.54	63.65	59.23	62.31

Table B.2: Comparison of MCR for English and Greek alphabet

Learning	Integration (output)	Integration (input)	English	Greek
0.0	0.0	57.0	0.96	5.83
0.0	0.0	114.0	3.65	5.83
0.0	0.0	228.0	14.42	13.33
0.0	0.0	341.0	43.27	23.33
0.0	2.0	0.0	0.00	1.67
0.0	2.0	57.0	1.35	3.33
0.0	4.0	0.0	2.69	3.33
0.0	4.0	114.0	4.62	5.83
0.0	4.0	228.0	17.50	15.83
0.0	8.0	0.0	8.27	10.00
0.0	8.0	114.0	8.27	11.67
0.0	12.0	0.0	17.12	18.33
2.0	0.0	0.0	17.88	23.33
2.0	0.0	57.0	20.19	24.17
2.0	0.0	114.0	14.62	21.67
2.0	2.0	0.0	21.92	22.50
2.0	2.0	57.0	19.42	21.67
2.0	4.0	0.0	21.35	20.00
4.0	0.0	0.0	31.92	33.33
4.0	0.0	57.0	23.08	30.83
4.0	0.0	114.0	27.31	31.67
4.0	0.0	228.0	32.50	26.67
4.0	0.0	341.0	54.81	29.17
4.0	2.0	0.0	28.85	32.50
4.0	2.0	57.0	28.46	34.17
4.0	4.0	0.0	25.96	35.00
4.0	4.0	114.0	28.27	33.33
4.0	4.0	228.0	32.31	30.83
4.0	8.0	0.0	37.50	39.17
4.0	8.0	114.0	32.88	40.83
4.0	12.0	0.0	40.19	46.67
8.0	0.0	0.0	40.77	43.33
8.0	0.0	114.0	42.31	38.33
8.0	0.0	228.0	44.42	43.33
8.0	0.0	341.0	58.27	46.67
8.0	4.0	0.0	41.54	52.50
8.0	4.0	114.0	43.27	50.83
8.0	4.0	228.0	46.92	50.00
8.0	8.0	0.0	50.19	55.83
8.0	8.0	114.0	50.00	46.67
8.0	12.0	0.0	52.50	50.00
12.0	0.0	0.0	53.85	54.17
12.0	0.0	114.0	52.88	59.17
12.0	0.0	228.0	53.65	64.17
12.0	0.0	341.0	67.69	57.50
12.0	4.0	0.0	53.46	52.50
12.0	4.0	114.0	54.81	53.33
12.0	4.0	228.0	57.31	60.83
12.0	8.0	0.0	57.12	57.50
12.0	8.0	114.0	57.12	58.33
12.0	12.0	0.0	63.65	68.33